XVT Design
Manual

xvt

© 2011 Providence Software, Inc. All rights reserved. Using XVT for Windows® and Mac OS

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Providence Software Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Providence Software Incorporated. Providence Software Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization. XVT, the XVT logo, XVT DSP, XVT DSC, and XVTnet are either registered trademarks or trademarks of Providence Software Incorporated in the United States and/or other countries.

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Macintosh is a trademark of Apple Inc. registered in the U.S. and other countries. All other trademarks are the property of their respective owners.

XVT-DESIGN

CONTENTS

1 INTRODUCTION

This chapter provides a brief overview of XVT-Design and suggests how to use this manual most effectively.

1.1. What is XVT-Design?

XVT-Design is a graphical, interactive design tool and application generator. It simplifies the design and implementation of graphical-user-interface-based applications in three major ways:

- XVT-Design lets you create the user-interface objects of your application (windows, controls, menus, and so on) graphically and interactively, rather than by programming manually.
- XVT-Design provides a TestMode that lets you preview your application's user interface *without* separate compilation and linking steps. You can use XVT-Design to build and refine application prototypes rapidly without writing any code.
- XVT-Design generates event handlers and other source code for your application's user interface. Instead of rewriting "generic" user-interface code by hand for each new application, you can use XVT-Design to create this code automatically.

Without XVT-Design, it is necessary to write resource language source code by hand to define the necessary GUI resources for an application in a portable manner. Since you cannot visually inspect your resources until you run the application, this is a cumbersome, iterative process.

With XVT-Design, you create a "project" file containing the GUI resources for your application. For each project, you can create any number of dialog boxes and windows. Then you draw the needed controls right on your computer screen, placing them just where you want within a window or dialog box—without ever having to calculate the screen coordinates numerically.

XVT-Design automatically generates a complete set of source files for your application:

- C code for the application's user interface
- A C header file containing function prototypes and constant declarations
- An XVT Universal Resource Language (URL) file, containing portable definitions of your application's resources
- A complete makefile for compiling and linking your application
- A help text file

1.2. Using This Manual

This manual is intended for programmers who have some experience with building and using GUI applications. It assumes familiarity with basic GUI concepts, such as events and resources, and some familiarity with the XVT Portability Toolkit. Since XVT-Design generates XVT-based programs, use this manual in conjunction with the other XVT Development Solution manuals.

Because XVT-Design is an XVT application available on multiple platforms, separate installation instructions are provided with each media distribution. This manual assumes you have already installed XVT-Design and the XVT Portability Toolkit on your development system.

Along with this introduction, this manual contains the following chapters:

Chapter 2: Concepts

Introduces the general ideas required to use XVT-Design well.

Chapter 3: Tutorial

Provides a step-by-step tutorial that introduces you to many of the main features of XVT-Design.

Chapter 4: Using XVT-Design

Describes how to use all the features of XVT-Design.

Chapter 5: Project File Management

Describes XVT-Design's support for multi-developer programming projects, through splitting and merging project files.

Chapter 6: Internationalization
Summarizes all the menu commands in XVT-Design and describes Control Description Files.

Chapter 7: Reference
Describes the issues relating to internationalizing your XVT application.

Appendix: Image Editor
Describes how to use the Image Editor.

1.2.1. On-line Help

In addition to this manual, summary documentation is available for all XVT-Design features and menu items in an on-line Help utility.

1.3. Conventions Used in This Manual

In this manual, the following typographic and code conventions indicate different types of information.

General Conventions

`code`
This typestyle is used for code and code elements (names of functions, data types and values, attributes, options, flags, events, and so on). It also is used for environment variables and commands.

bold
Bold type is used for filenames, directory names, and program names (utilities, compilers, and other executable).

italics
Italics are used for emphasis and the names of documents.

Tip: This symbol marks the beginning of a procedure having one or more steps.

Note: An italic heading like this marks a standard kind of information: a Note, Caution, Example, Tip, or See Also (cross-reference).

Code Conventions

`<non-literal element>` or `non_literal_element`
Angle brackets or italics indicate a non-literal element, for which you would type a substitute.

[optional element]
: Square brackets indicate an optional element.

2

XVT-DESIGN CONCEPTS

This chapter briefly introduces some terms and concepts that you'll need before working through the tutorial in the following chapter.

2.1. GUI Objects

A graphical user interface (GUI) has four main types of graphical components: windows, dialogs, controls, and menus.

You can lay out all these GUI objects with XVT-Design. XVT-Design also constructs the C-language source code to manage the objects.

2.2. Portable Resources

XVT-Design transforms your WYSIWYG layout of GUI objects into a portable resource language called URL. Since the URL code is portable, you need only use XVT-Design to generate it once. You can compile the URL code to any supported native resource format.

XVT's **curl** compiler translates URL resource specifications into the native resource language. The native resources are bound to the executable program, either by loading the resources directly into the executable image or by creating a resource file with a canonical name. (Native environments use differing methods to bind resources.)

See Also: For more information, see *XVT Portability Toolkit Guide*, Chapter 5.

2.3. Events

XVT bases its Portability Toolkit on a set of abstract, portable event representations. These deliver user and GUI system event data to GUI objects within your application.

In XVT-Design, each type of event is represented by a tag in the Action Code Editor. Developers generally use the Action Code Editor to enter calls to functions (defined in external source files), which are executed in response to particular events.

See Also: For more information, see *XVT Portability Toolkit Guide*, Chapter 4.

2.4. Event Handlers

An event handler is a function with the proper prototype for receiving events, meaning that it accepts a WINDOW and an EVENT* as arguments.

Most windows—and all dialogs—must be assigned an event handler to process the events generated during their lives. The exception is the screen window, which has no event handler because it receives no events. Windows can have unique event handlers, or multiple windows can share a common event handler.

2.5. GUI Object Attributes

All the GUI objects provided by the XVT Portability Toolkit have a number of attributes that describe their appearance and behavior. For example, windows might be sizeable or iconizable, or might contain scrollbars or titlebars.

XVT-Design allows you to set the values of GUI objects interactively, rather than specifying them programmatically.

See Also: For more information, see *XVT Portability Toolkit Guide*.

2.5.1. Geometry

Rectangles specify the size and position of windows, dialogs, and controls. Instead of describing the size and position of objects in resource-description text files, XVT-Design lets you create and modify objects graphically.

2.5.2. Title

All objects have a title string, which is the name of the object from the application user's point of view. The object may or may not have a visible title, depending on the type and conventions of the native windowing system. (For instance, document windows and buttons almost always have visible titles, but scrollbars do not.)

2.5.3. Symbolic Identifier

Objects have a second string that is used by the application developer, rather than the user. This symbolic identifier string lets you refer to the object with a symbolic name, instead of its resource ID number. The symbolic identifier string associates a symbolic name with the ID number that is used in the resource file.

2.5.4. Other Attributes

Each type of object has additional attributes, specific to its type. For instance, windows have attributes that describe their border decorations, and menus have attributes that specify accelerator and mnemonic keys.

2.6. User Interface Code

In addition to creating resources graphically, XVT-Design helps you create the program code for your application's user interface objects.

2.6.1. Integrated Code Editing

XVT-Design's Action Code Editor (ACE) lets you create and edit your user interface source code without leaving XVT-Design. This ensures that the code you enter is preserved as part of the interface definition.

Tip: XVT recommends putting only small fragments of source code into your project with the Action Code Editor and putting the bulk of your user-written code in external files.

2.6.2. Structural Code

XVT-Design creates event-handling functions and other code that produces a structure for your application's user interface. Rather than writing your event handlers and other "generic" code from scratch, you can use XVT-Design to generate this code for you.

Note: While using XVT-Design, you do not actually see the structural code. The code you create is integrated with the structural code when XVT-Design generates the source code files for your application.

2.6.3. Tags

XVT-Design assigns two types of tags to your user interface objects: event tags and special tags.

Event tags
All user interface objects have a number of associated events, that is, occurrences that the application responds to. XVT-Design assigns *event tags* to these events. The event tags correspond exactly to the `E_*` events defined in the XVT API. Each object also has a number of special tags.

Special tags
Special tags do not correspond to any runtime events, but are "markers" in the generated code for an object. They indicate positions in the framework where you may want to insert your own code. For instance, the **Var Decl** special tag marks a location appropriate for declaring local variables for an object's event handler.

2.6.4. Action Code

Action code is C source code that implements some action in response to an event.

You add action code to your application with the Action Code Editor. XVT-Design incorporates it into structural code when generating your application's source code files.

2.6.5. Context

Action code is always associated with a specific context. In most cases, the context represents an event generated by a specific action or specific user interface object. (If the context includes a special tag, it represents a place for you to insert code *not* related to an event, for example allocating or releasing dynamic memory.)

A context is composed of three parts:

Module
A module is a user-interface component that contains other components. Windows, dialogs, menubars, and the application itself are all modules.

Object
An object is one of the components contained by a module. Controls and menu items are objects. In XVT-Design, modules

are also considered to be objects (in a sense, they contain themselves).

Tag

Every object has one or more tags (event or special), as described above.

The unique combination of a module, an object, and a tag makes up a context. Three list buttons in the Action Code Editor specify the context for action code.

2.7. TestMode

XVT-Design's TestMode lets you view an application's user interface without compiling and linking source code. As a result, you can rapidly refine the appearance of user interface objects without leaving XVT-Design.

2.8. Connections

Rather than interpreting or executing an application's source code, XVT-Design's TestMode defines relationships between objects by using connections.

Like action code, connections are associated with tags. A connection opens or closes a container (window or dialog) when a tag's event occurs. Connections can also invoke XVT's predefined dialogs, such as the standard open-file and save-file dialogs.

2.9. External Source Code

You can include other source code modules into an XVT application by adding them to your XVT-Design project. To do this, choose External Files from the File menu. XVT-Design includes any files specified in this way when it generates the makefile.

Tip: XVT recommends the following approach to adding your code to the default C-language structural code supplied by XVT-Design:

1. Write functions in external files to hold any large blocks of action coder.
2. In the Action Code Editor, place calls to the functions that you defined in external files.
3. Then, using the External Files option from the File menu, tell XVT-Design to include the external files in the makefile.

XVT recommends this approach for several reasons:

- It supports modularity
- It keeps the code generated by XVT-Design simple and free of errors
- It keeps the code that is more apt to be changed and added to during the application's development *external* to the XVT-Design project
- It lets you edit external code without modifying the XVT-Design project file

Note: With XVT-Design, you can edit the generated files using a text editor and later recover code from inside of XVT-Design.

If you check the Code Recovery checkbox in the Project Attributes dialog, code fragments are enclosed between special comments in the generated code. You can edit the code between the comments using a standard text editor. Then click on a "Recover Code" button to recover all the changes you have made to the generated files.

2.10. XVT-Design Files

Your XVT-Design project file—a portable, binary file—contains the layout, GUI objects, action code, and configurations made within XVT-Design. In addition, XVT-Design generates the following files:

Module header and source files
XVT-Design generates source (**.c**) files and a header (**.h**) file for the application module (task window), and for each window, menubar, and dialog in the project.

Makefile
XVT-Design generates an application makefile, using a template appropriate for the platform/compiler. You select the template from a list in the Generate Application dialog.

Universal Resource Language (URL) file
This file defines the external resources of the project.

You can inspect these files, or change their names, by choosing Generate Application from the File menu.

Note: The project file does not display when you choose Generate Application. The prefix of the project filename matches the name of the project, but you can configure its suffix in XVT-Design's configuration file.

Figure 2.1 illustrates the relationship among all the XVT-Design files.

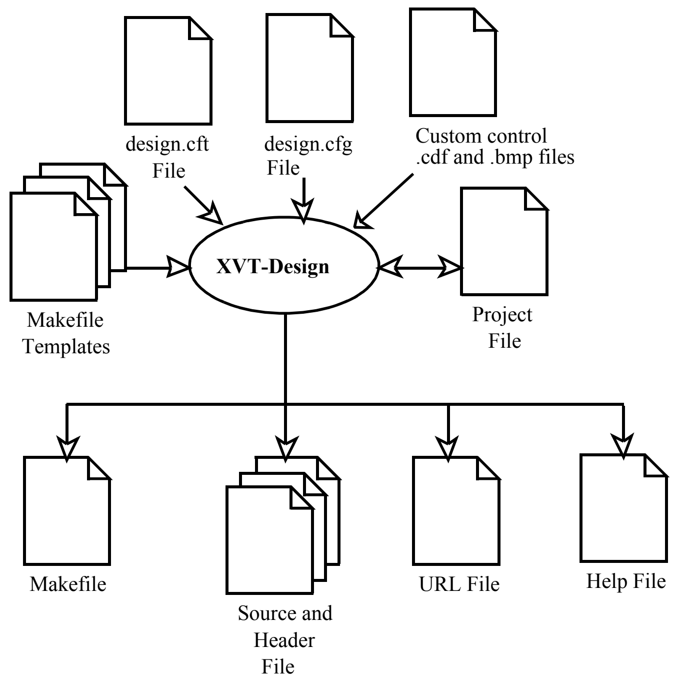


Figure 2.1.XVT-Design files

3

TUTORIAL

This tutorial chapter demonstrates how to use XVT-Design to build a sample application. The tutorial is designed to show you how quickly you can develop GUI applications with XVT's Development Solution for C.

Before doing the tutorial, you might read the previous chapter of this manual. It contains definitions of key terms used throughout the tutorial.

Tip: To begin the tutorial

Start XVT-Design.

3.1. The Hello Application

The sample application you will build is called "Hello." It has the following features:

- The user can open any number of windows by choosing the New menu item on the File menu. These windows display a message, chosen by the user, and can be moved, resized, and closed in the usual fashion of the native window system.
- The messages displayed in the windows can be changed by choosing menu items on the Choices menu. Choosing the Other Choices menu item brings up a dialog that has several other message options.
- The user can select the font, size, and style for the message using the Font menu.

Note: This is a hands-on tutorial. If you do not want to create the Hello application yourself, you will find a copy already created for you. Look for the **hello.dpr** file in the **tutorial** directory in your XVT directory.

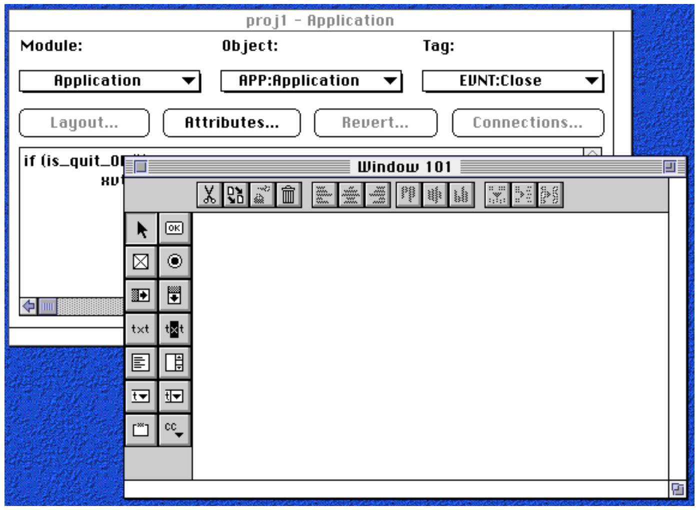
3.2. Creating a New Project

In building the tutorial application, as in building any application with XVT-Design, you first create a new project file for the application. An XVT-Design project contains most of the resources and source code for your application's user interface.

Tip: To create a new project:

1. From the File menu, choose New Project.

XVT-Design opens an Action Code Editor (ACE) window and a layout window, "Window 101."



```
if (is_quit_O
        xv
```

Figure 3.1.The Action Code Editor and a layout window (Macintosh platform)

You won't use either the Action Code Editor or the layout window right away.

2. Close both windows by clicking their close boxes.

3.3. Creating a Menubar and Menus

Now that the project file is open, you will create the menubar and menus for your application. The finished menubar looks like this:

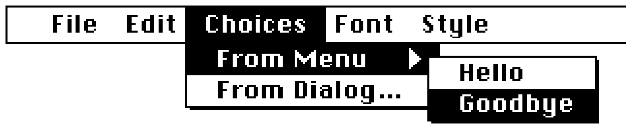


Figure 3.2.The Menubar for the sample application

XVT-Design supplies the File, Edit, Font/Style, and Help menus—they're called the *standard menus*. (The Help menu doesn't appear explicitly on all platforms.) You will create the Choices menu and its submenu.

To create this menubar, in the next sections you will follow these basic steps:

- Create a new menubar
- Create the Choices menu
- Add items to the Choices menu
- Create the submenu
- Add items to the submenu

3.3.1. The Menubar Editor

Tip: To create the new menubar

1. From the Tools menu, choose Menubar Editor.

The Menubar Editor looks like this:

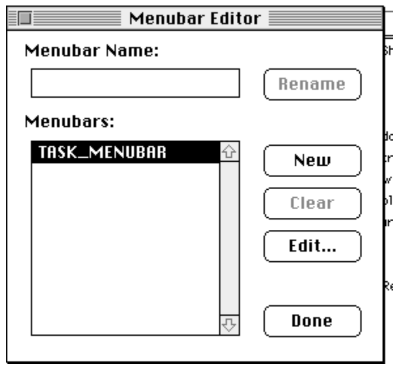


Figure 3.3.The Menubar Editor

The list box in the Menubar Editor shows all the menubars in the project. Notice that XVT-Design has automatically created one menubar, called TASK_MENUBAR. TASK_MENUBAR is the default menubar containing the four standard menus mentioned in the previous section.

3.3.1.1. Creating a New Menubar

To create a new menubar,

2. Click New

XVT-Design adds a new menubar to the list box, with a default name of MENU_BAR_2.

Change the name to WIN_MENUBAR:

3. Enter the name "WIN_MENUBAR" in the edit field at the top of the Menubar Editor.
4. Click Rename.

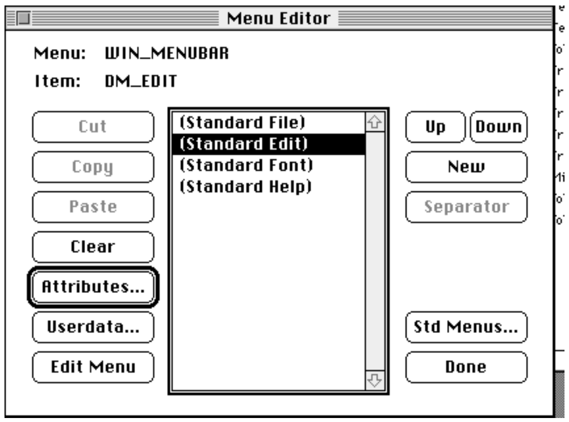
3.3.2. The Menu Editor

Now you will add a new menu—the Choices menu—to the menubar you created, using XVT-Design's Menu Editor.

Tip: To add a menu to a menubar,

1. In the list box, select your menubar, WIN_MENUBAR.
2. Click Edit in the Menubar Editor to bring up the Menu Editor.

The Menu Editor looks like this:



Menu: WIN_MENUBAR
Item: DM_EDIT

(Standard File)
(Standard Edit)
(Standard Font)
(Standard Help)

Figure 3.4.The Menu Editor

The name of the menubar appears at the top of the Menu Editor, to show which menu you are editing. The list box in the Menu Editor lists the names of the menus on the menubar. The menu items will read left-to-right on the menubar, in the same order as they appear reading top-to-bottom in the list box. XVT-Design has already added the standard menus to the menubar; they are enclosed in parentheses.

In the Menu Editor, the New button creates new menus, and adds them to the list.

3.3.2.1. Creating a New Menu

Tip: To make the Choices menu appear to the right of the Edit menu on the menubar, insert it in the list after the Standard Edit menu.

3. Select Standard Edit in the list box, and click New.

A new menu appears in the list, with the default title “new item.”

3.3.2.2. Changing the Menu Title

One of the attributes of a menu is its title. Instead of “new item,” you’ll change the title to “Choices.”

4. Click Attributes in the Menu Editor or double-click on Standard Edit in the listbox.

The attributes dialog for the new menu appears. The title appears at the top of the dialog.

5. Click in the edit control and change the title to “Choices.”

After you’ve changed the title, the dialog looks like this:

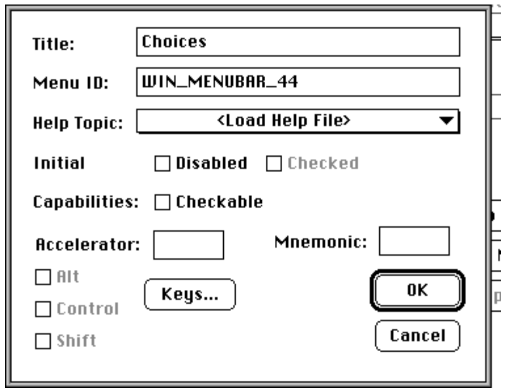

Title: Choices
Menu ID: WIN_MENUBAR_44
Help Topic: <Load Help File>
Initial ☐ Disabled ☐ Checked
Capabilities: ☐ Checkable
Accelerator:
Mnemonic:
☐ Alt
☐ Control
☐ Shift

Figure 3.5.The Menu Attributes dialog

Notice that you could use this dialog to set other attributes, such as the menu's Menu ID (symbolic identifier) string. For this menu, leave those attributes unchanged.

6. Click OK to save the title and close the attributes dialog.

3.3.2.3. Adding Items to the Menu

You have created a new menubar, and added a menu to it. Now you will add items to the menu. For the Choices menu, these items are "From Menu" (which has a submenu) and "From Dialog."

7. In the Menu Editor, click Add Menu.

This opens another Menu Editor window, in which you can add the items to the Choices menu. The list of menu items is empty, since no items have been added to the menu yet.

Creating the First Item

8. To create the first item, click New.

A new item is added to the list, as shown in Figure 3.6.

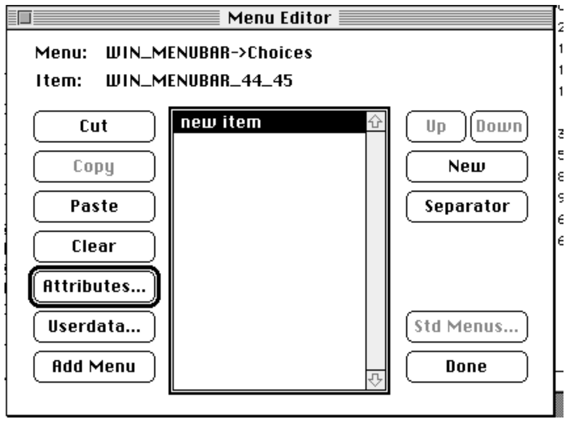
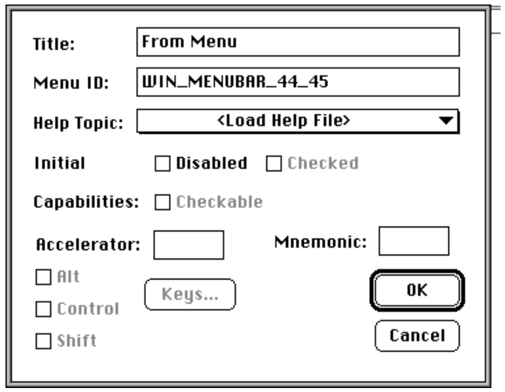


Figure 3.6.The Menu Editor showing a new item

Change the title of the new item, just as you did for the Choices menu itself.

9. Click Attributes to open the attributes dialog for the new item.
10. In the title field, change the title to "From Menu."


Title: From Menu

Menu ID: WIN_MENUBAR_44_45

Help Topic: <Load Help File>

Figure 3.7.Changing the menu title to "From Menu"

11. Click OK to dismiss the attributes dialog.

Creating the Second Item

12. Create another menu item, using the same procedure you followed to create the first item.

13. Give it the title “From Dialog…”

Caution: Be careful to create a new item first, and *then* click on Attributes.

3.3.2.4. Associating Help Topics with Menu Items

XVT-Design lets you provide help text that is specific to the context the user has questions about. The “Menu Attributes” dialog contains a list box that lists all available help topics. If a help file were available, you would choose the topic that pertains to this particular menu item.

This tutorial does not address generating the help topics. Typically, you develop your application’s user interface before writing the help text. (Sometimes these tasks are done simultaneously). Once you have created a help file, you use XVT-Design to select a topic that addresses each item for which you want to provide help.

3.3.2.5. Creating a Submenu

The final addition to the menubar is the submenu for the From Menu item. This submenu has two items, “Hello” and “Good-bye.” A check mark appears next to these items when they are chosen. You build the submenu by adding items to the From Menu item, and setting their attributes, just as you did before.

Creating the First Submenu Item

14. In the Menu Editor’s list box, select the From Menu item.
15. Click Add Menu to open another Menu Editor window.
16. Add a new item, as before, by clicking New.

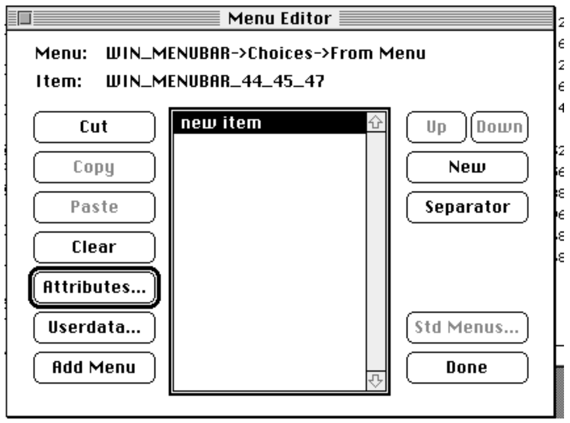


Figure 3.8.Adding a submenu to the “From Menu” menu

17. Next, click Attributes to bring up an attributes dialog box for the item.
18. Enter “Hello” in the Title field and “M_HELLO” in the Menu ID (symbolic identifier) field.

Your application’s source code will refer to the menu item by the symbolic identifier string, rather than using its equivalent integer resource ID.

19. Also click Checkable and Checked so the item will be initially checked.

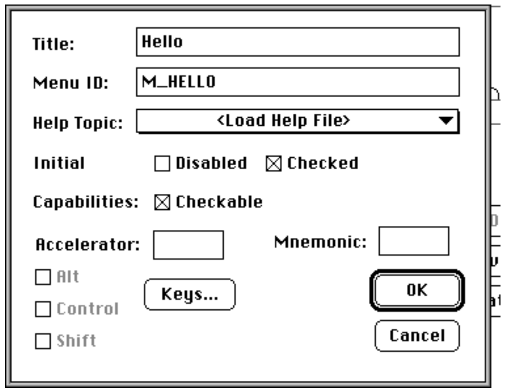
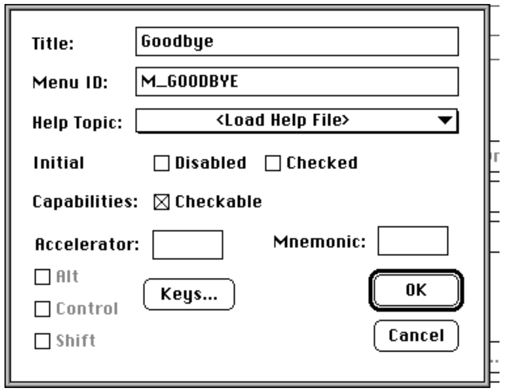

Title: Hello
Menu ID: M_HELLO
Help Topic: <Load Help File>
Initial: ☐ Disabled ☒ Checked
Capabilities: ☒ Checkable

Figure 3.9.Changing the menu item title to “Hello”

20. Click OK to dismiss the attributes dialog.

Creating the Second Submenu Item

21. Create a second menu item called "Goodbye," just as you created the Hello menu item.
22. Set its Menu ID (symbolic identifier) string to "M_GOODBYE".
23. And then click Checkable.



Title: Goodbye
Menu ID: M_GOODBYE
Help Topic: <Load Help File>
Initial ☐ Disabled ☐ Checked
Capabilities: ☒ Checkable
Accelerator:
Mnemonic:
☐ Alt
☐ Control
☐ Shift

Figure 3.10.Changing the second menu item title to "Goodbye"

You have completed creating the menus and menu items for the application.

24. Close all the Menu Editor windows by clicking Done in each of the three Menu Editor windows.
25. In the initial Menubar Editor dialog, click Done again.

3.4. Saving the Project

Tip: This is a good time to save the work you've done so far.

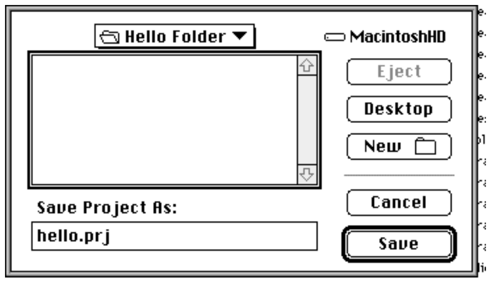
1. Choose Save Project from the File menu.

Since this is the first time you've saved this project, a standard file save dialog appears.

XVT-Design (running on the Mac and Windows platforms) gives the project file a default name of "**proj1.dpr**."

2. Change the default name to "**hello.dpr**."
3. Use the Save As dialog from the File menu to save the file to the directory you want.

4. Click Save.



Save Project As:

hello.prj

Figure 3.11.Saving the "hello.dpr" project

As with any application, it's a good practice to save your file frequently.

3.5. Creating Containers

Next you will create the containers—windows and dialogs—for the application. You need three containers:

- The document window for the application, to display a message chosen by the user
- A Choices dialog, with several controls, to allow the user to choose one of several messages
- An "About box" dialog, to display the name of the application

Each of these containers has several attributes, such as its name and size, that you will set. First you will create the document window and set its attributes. Then you will create the Choices dialog and its controls. Finally, you will create the About box.

3.5.1. Creating the Message Window

At start-up, XVT-Design created an empty layout window called Window 101, which we'll now use as our message window.

Tip: To create a message window,

1. From the Tools menu, choose Action Code Editor.
2. Select Window 101 from the Module list button
3. Click Layout to open the window.
4. Close the ACE window by clicking its close box.

If you wanted to create additional windows, you could choose New Window from the Window menu. When you do this, XVT-Design creates a new window resource and opens a layout window for it.

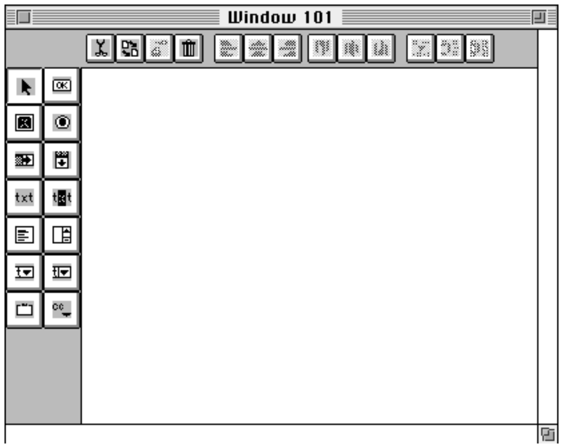


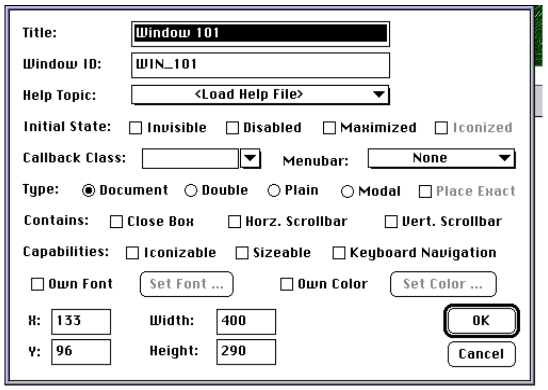
Figure 3.12.Layout window for Window 101

3.5.1.1. Setting the Window's Attributes

5. Choose Attributes from the Edit menu to open the attributes dialog for the new window.

(As a shortcut, you can double-click in the client area of the window to open its attributes dialog.)

This dialog lets you change all of the attributes of the window, such as its title, size and location, and border style.

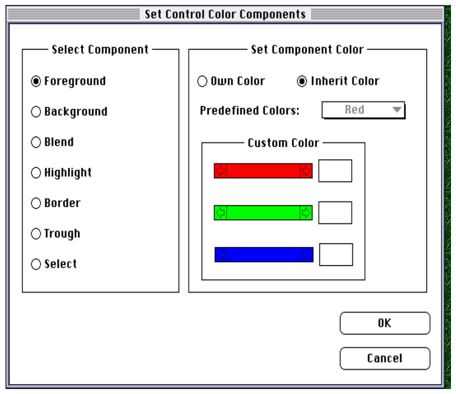


Title: Window 101
Window ID: WIN_101
Help Topic: <Load Help File>
Initial State: ☐ Invisible ☐ Disabled ☐ Maximized ☐ Iconized
Callback Class: Menubar: None
Type: ● Document ○ Double ○ Plain ○ Modal ☐ Place Exact
Contains: ☐ Close Box ☐ Horz. Scrollbar ☐ Vert. Scrollbar
Capabilities: ☐ Iconizable ☐ Sizeable ☐ Keyboard Navigation
☐ Own Font Set Font ... ☐ Own Color Set Color ...
X: 133 Width: 400
Y: 96 Height: 290

Figure 3.13.Setting attributes for a new window

XVT-Design gave the window a default title of "Window 101."

6. Change it to "Message" by clicking in the Title edit control and editing the string.
7. Change the Window ID (symbolic identifier) string from the default "WIN_101" to "WIN_MESSAGE."
8. Click the check boxes labeled Close Box and Sizeable.
9. Click on Own Color check box.
10. Click on Set Color push button.
11. Click on Own Color radio button. A dialog will appear. See Figure 3.14. below.
12. Click on the Select Component button.
13. Choose Magenta from the Predefined color list.



Set Control Color Components

Select Component: [✓ Foreground] Background Blend Highlight Border Trough Select

Set Component Color: Own Color [✓ Inherit Color]

Predefined Colors: Red

Custom Color

OK

Cancel

Figure 3.14.Setting control color components

14. Click OK.

3.5.1.2. Associating the Window's Menubar

You have already created a special menubar for the message window: the menubar entitled WIN_MENUBAR. Now you will associate this menubar with the application's message window.

15. Click on the list button located to the right of the label Menubar.

A list of all the menubars in your project descends from the list button.

16. To associate your previously created menubar with the message window, choose WIN_MENUBAR from the menubar list button.
17. Click OK to close this attributes dialog.

3.5.1.3. The Object Palette

When you create a new window or dialog in XVT-Design, or when you open an existing window or dialog, it contains an "Object Palette" on the left side. This palette contains graphical toggle buttons for all the controls you can place into the window or dialog. Figure 3.15. shows the toggle buttons and the controls they represent.

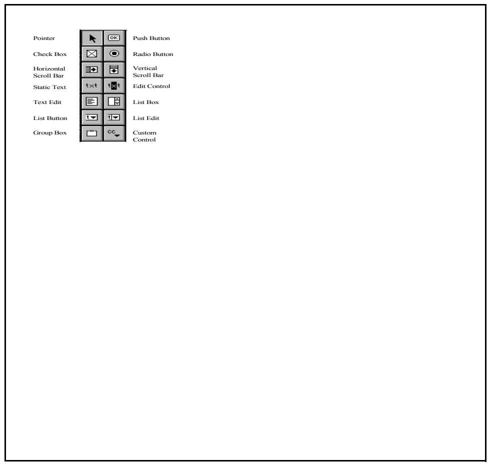


Figure 3.15.The object palette

Note: After you place the controls into your window or dialog, you can hide the object palette so you can more accurately see what the window or dialog looks like.

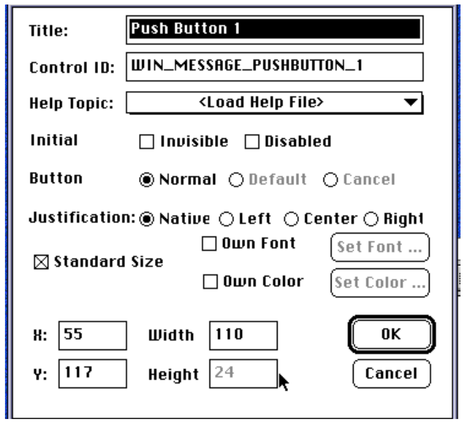
To hide the palette, select Hide Object Palette from the Layout menu.

To redisplay it, select Show Object Palette.

3.5.1.4. Adding a Push Button Control

The message window in your sample application will have one control: a push button.

18. To create the button, click the push button toggle button in the object palette.
19. Then click in the lower section of the Message window to place the button. (An alternate method is to choose Push Button from the Controls menu and click in the window.)
20. Select the pointer toggle button from the object palette and double-click on the push button to open its attributes dialog.


Title: Push Button 1

Control ID: WIN_MESSAGE_PUSHBUTTON_1

Help Topic: <Load Help File>

Initial: ☐ Invisible ☐ Disabled

Button: [✓ Normal] Default Cancel

Justification: [✓ Native] Left Center Right

☒ Standard Size ☐ Own Font ☐ Own Color

X: 55 Width 110

Y: 117 Height 24

Figure 3.16.Setting attributes for a push button

The default title of the push button is “Push Button 1”.

21. Change it to “Custom String...”

22. Click OK to close the dialog.

You may find that the button is too small to contain the new name. If so, enlarge it by dragging the small rectangle near its lower-right corner. When you’re done, the window should look like this:

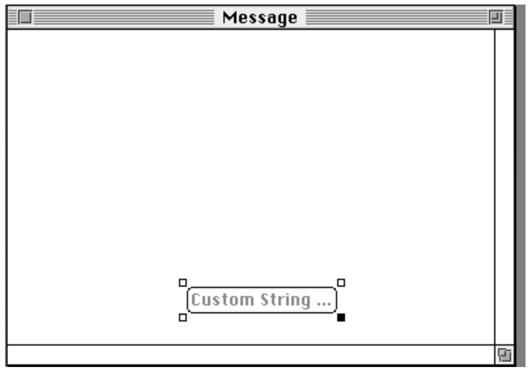


Figure 3.17.How the push button looks in the window

3.5.1.5. Saving the Project

Tip: Again, this might be a good time to save the work you’ve done so far.

1. Choose Save Project from the File menu.

3.5.2. Creating the Other Choices Dialog

Now you will create the Other Choices dialog. First, you'll create an empty dialog and set its attributes. Then, you'll place some controls in the dialog, and set their attributes.

3.5.2.1. Creating a New Dialog

Tip: To create the dialog

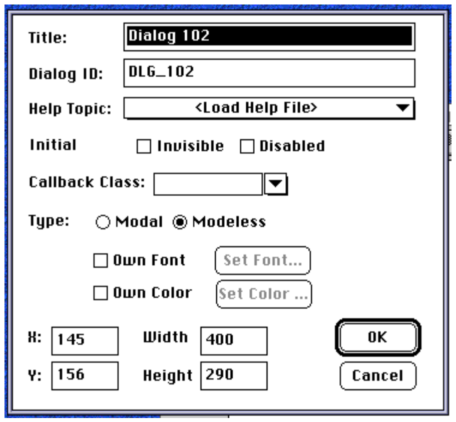
1. Choose New Dialog from the Window menu.

XVT-Design opens a layout window for the dialog.

Note: XVT-Design uses resizeable windows to represent both windows and dialogs, so that you can easily change the size and position of the dialogs you create. When your finished application is running, an actual native dialog is used appropriately.

3.5.2.2. Setting the Dialog's Attributes

2. Double-click in the dialog layout window to open its attributes dialog:



Title: Dialog 102
Dialog ID: DLG_102
Help Topic: <Load Help File>
Initial ☐ Invisible ☐ Disabled
Callback Class:
Type: ○ Modal [✓ Modeless]
☐ Own Font
☐ Own Color
X: 145 Width 400
Y: 156 Height 290

Figure 3.18.Setting attributes for a dialog

3. Change the dialog's title to "Other Choices" and change its Dialog ID (symbolic identifier) string to "DLG_CHOICES."

Note: The following step is an important one; make sure you perform it.

4. Click Modal to make the Choices dialog modal.
5. Click OK to close the attributes dialog.

3.5.2.3. Adding Radio Buttons

Now you'll add a group of radio buttons to the dialog. XVT-Design provides two methods for creating multiple controls:

Method 1: Using the object palette

1. From the object palette on the left side of the dialog, select the desired control by its toggle button.
2. Click in a layout window to create a control.
   Every time you click in the layout window, a control of this type is created.
3. When you are done placing controls of this type, choose the pointer toggle button from the object palette.

Method 2: Using the Controls Menu

1. Before choosing the desired control from the Controls menu, press and hold the Shift key on your keyboard. Then choose the control.
2. Click in a layout window to create a control.
   Every time you click in the layout window, a control of this type is created.
3. When you're done creating controls of this type, choose Pointer (or another control) from the Controls menu.

Tip: Use one of these methods to place four radio buttons in the Other Choices dialog.

When you're finished creating controls, remember to choose the pointer toggle button from the object palette or Pointer from the Controls menu.

The dialog's layout window should look like Figure 3.19.

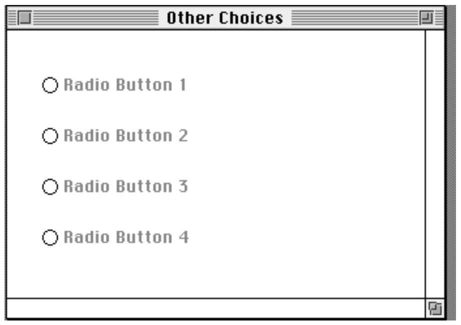


Figure 3.19.Adding radio buttons to the dialog

3.5.2.4. Changing the Radio Button Titles

Now change the titles of the radio buttons to correspond to messages that the user of the application can choose.

1. Double-click the first radio button (or choose Attributes from the Edit menu) to bring up its attributes dialog.
2. Change its title to “Have a nice day!” and click OK to dismiss the attributes dialog.
3. Change the titles of the remaining radio buttons to “See ya later, alligator!”, “Beam me up, Scotty!” and “Make it so!” (or whatever other messages strike your fancy).

Note: In the layout window, the titles of the buttons might be truncated on the right; if so, make the buttons larger by clicking them and dragging the black rectangle at their lower-right corner. You might also need to change the size of the dialog itself to accommodate the controls; do this by resizing the dialog’s layout window.

3.5.2.5. Change the Radio Button Fonts

For each of the radio buttons, do the following:

1. Double-click on the individual radio buttons to bring up the Attributes window.
2. Click on the Own Font check box.
3. Click on the Set Font button.
4. Change the fonts and styles to ones of your own choosing.
5. Click OK when you are done.

6. Click OK to dismiss the Radio Button Attributes editor window.

3.5.2.6. Adding Push Buttons

Tip: Now you will give the dialog two push buttons: OK and Cancel.

Creating the OK Button

1. Create a push button, and double-click it to open its attributes dialog.
2. Change its title to “OK,” and click Default.

In a dialog, the button having the Default attribute appears with a thick border drawn around it (at *runtime*, not in the Layout Window). It recognizes Return and Enter keystrokes. (For example, in the attributes dialog, OK is the default button.)

Creating the Cancel Button

3. Create a second push button, and double-click it to open its attributes dialog.
4. Change its title to “Cancel,” and click the Cancel *radio* button. (You can change the font for the button at this point, too.)

In a dialog, the button having the Cancel attribute has the behavior appropriate for the native platform. (On most platforms this makes the “Escape” key equivalent to the Cancel button.)

5. When you have finished setting the title of the control, click OK to quit the attributes dialog.

The dialog should now look something like this:

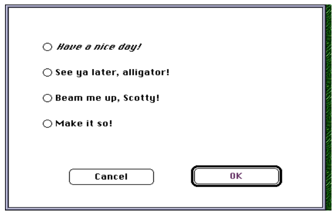


Figure 3.20.How controls look in the new dialog

3.5.2.7. Using Layout Options

The Layout menu contains several items for aligning, spacing, and sizing controls within a dialog or window. The toolbar that appears at the top of the window or dialog in layout mode provides another way to access these layout functions—with toggle buttons

1. Select all four of the radio buttons by clicking in the upper-left corner of the window and holding the mouse button down as you drag a rectangle to enclose all four buttons.
2. Align the left edge of the four radio buttons by using either the Align Left menu option or toggle button.
3. While all buttons are still selected, choose the Even Vertical Spacing menu option or toggle button.

   You can also position the group of radio buttons and move them together within the window by clicking on any one of the radio buttons and holding the mouse button down as you drag the group.
4. Use the alignment functions (from the Layout menu or toolbar) to position the two push buttons in the dialog.
5. Then, to get an accurate picture of what your dialog looks like, hide the toolbar and object palette (by selecting these items from the Layout menu).
6. Do any final positioning of objects with the toolbar and object palette hidden.

3.5.3. Creating an About Hello Dialog

The last container to create is the About box. This simple dialog contains some text and one push button.

Tip: To create the dialog,

1. Choose New Dialog from the Window menu.
2. Open its attributes dialog, change its title to "About Hello," its Dialog ID (symbolic identifier) string to "DLG_ABOUT," and click Modal. Click OK to dismiss the dialog.
3. Next, add two static text controls and a push button to the About dialog.
4. Change the title of the first static text control to "Hello version 1.0," and the title of the second to "A simple application created with XVT-Design."

   (Adjust the size of the control and the size of the dialog as needed so this entire string is visible.)
5. Change the title of the push button to OK, and check its Default check box.

When you're finished, the dialog should look something like this:

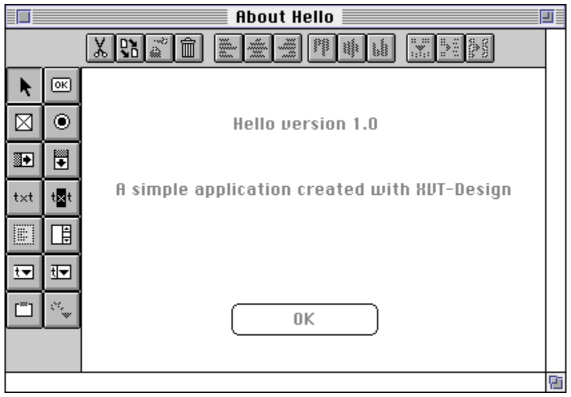


Figure 3.21.An About box dialog

To get an accurate picture of what your dialog looks like,

6. Using the options from the Layout menu, turn off the toolbar and object palette.

7. Do any final positioning of objects with the toolbar and object palette hidden.

3.6. Setting Application Attributes

So far you have been creating the individual user-interface objects of the Hello application, and setting their attributes.

Tip: Now you will set the attributes of the application itself.

1. Choose Project Attributes from the Edit menu to open the Project attributes dialog.

In the Project Attributes dialog, you will set the following seven attributes:

Task Menubar
The Task Menubar is the menubar attached to the task window. A list box shows the names of all the menubars in the project. The selected name is the task window's menubar.

2. For your Hello application, this should be TASK_MENUBAR.

About Box
The About Box is the dialog displayed when the application user brings up the About box. A list box shows the names of all the dialogs in the project. The selected name is the About box.

3. Click About Hello, the name of the dialog you created previously.

Task Window Title
This is the title of the task window. (This feature is not applicable on the Mac platform.) The title is displayed in an edit control.

4. Change it to "XVT-Design Tutorial."

Document Prefix
The Document Prefix is a string put at the beginning of the titles of the application's document windows. It is displayed in an edit control.

5. Change the Document Prefix to "Hello."

Internationalization
This check box allows you to generate internationalized code from XVT-Design.

Code Recovery
This check box must be clicked *before* you begin generating the

application in order to enable code recovery. See Chapter 4 for details.

6. The check box must remain selected to allow recovery of code *after you have generated the application.*

Own Font and Own Color
These buttons allow you to change the default control fonts and colors *for the entire application*. If you want all the control, containers, etc., to be , for example, magenta in color and in courier font, click one or both of these buttons.

Note: Individual GUI objects, windows, etc., can have their colors and fonts set as well. Settings at the individual level will override any settings made at the application level. These buttons allow you to change the default control settings for the *entire application.*

When you have finished setting the project attributes, the dialog should look like this:

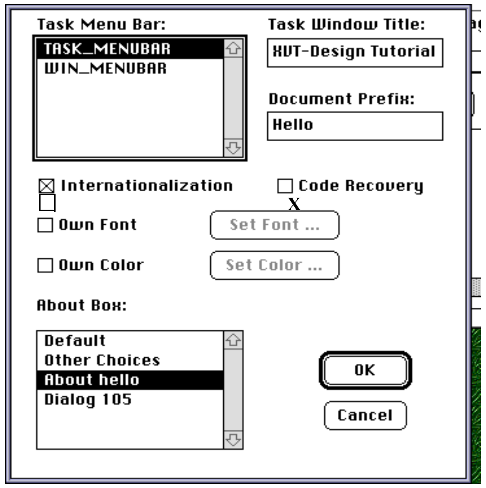

Task Menu Bar: TASK_MENUBAR (selected), WIN_MENUBAR
Task Window Title: XVT-Design Tutorial
Document Prefix: Hello
☒ Internationalization
☐ Code Recovery
☐ Own Font
☐ Own Color
About Box: Default, Other Choices, About hello (selected), Dialog 105

Figure 3.22.The Project Attributes dialog

7. Click OK to save these changes.

3.7. Setting Connections Between Objects

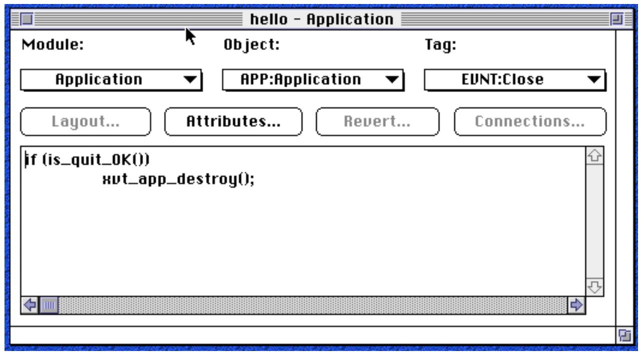
Now that you have built all the user-interface objects for the Hello application, you will begin to build the program that makes these objects do something.

If you were creating this application without XVT-Design, you would start writing C source code at this point. However, with connections and TestMode, XVT-Design lets you develop much of the functionality of your application's user interface *without* writing any code.

Tip: To start building the application's source code, you'll create connections by using XVT-Design's Action Code Editor.

1. Close any layout windows that you have left open, and choose Action Code Editor from the Tools menu. (You can leave layout windows open if you want, but your screen may not match the following illustrations if you do so.)

The Action Code Editor looks like this:



```
if (is_quit_OK())
        xvt_app_destroy();
```

Figure 3.23.The Action Code Editor

The ACE Context

Look at the three list buttons near the top of the ACE window, labeled Module, Object, and Tag. The settings of these list buttons constitute the *context* of the ACE—the unique combination of a module, one of the objects contained by the module, and one of the tags for the object.

The text-editing pane in the ACE always displays the action code for the context shown by the list buttons.

3.7.1. Task Menubar Connections

When the user chooses New from the File menu of the Task menubar, a new window will be created. You will set a connection for the Select tag of this menu item that opens the Message window.

To create this connection,

2. First set the context of the Action Code Editor as follows:
   - Set the Module list button to TASK_MENUBAR
   - Set the Object list button to ITEM:New

The list buttons in the ACE should look like this:

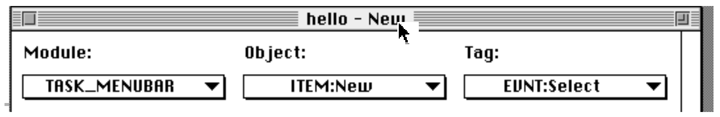


Figure 3.24.Setting the context of the ACE

In the remainder of this tutorial, the following format will represent the Action Code Editor's context:

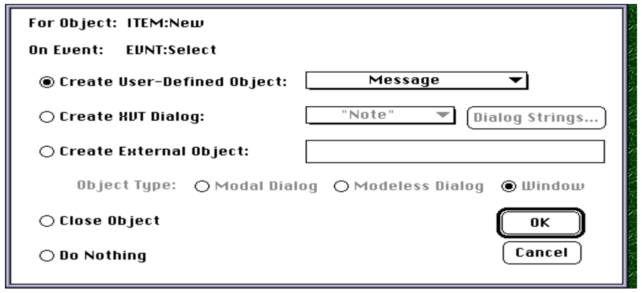
TASK_MENUBAR	ITEM:New	EVNT:Select

3. Click Connections ...to open the Connections dialog.
4. Since this connection will open one of the containers you have constructed, click Create User-defined Object.

A list button next to this radio button lists all the containers in the project.

5. Set this list button to Message, the title of the application's window.

The Connections dialog looks like Figure 3.25.



For Object: ITEM:New

On Event: EVNT:Select

[✓ Create User-Defined Object: Message]

Create XVT Dialog: "Note" Dialog Strings...

Create External Object:

Object Type: Modal Dialog Modeless Dialog [✓ Window]

Close Object

Do Nothing

Figure 3.25.The Connections dialog

6. Click OK to dismiss the dialog.

Notice that XVT-Design adds the following code to the ACE’s text-editing pane:

```
if (!xvt_win_create_res(WIN_MESSAGE, TASK_WIN,
        EM_ALL,WIN_MESSAGE_eh, 0L))
        xvt_dm_post_error("Can’t open window");
```

When executed, this code invokes your window. It checks the return value of xvt_win_create_res for errors, and puts up an error message if the window is not successfully created.

What Happens When You Create a Connection

When you create a connection, XVT-Design does two things:

- Records the connection in the project file, so that the connection will occur during TestMode.
- Adds action code to the ACE’s editing pane. This code is added to the module’s source code file when XVT-Design generates your application’s files, so that the connection will occur when the compiled application is executed.

Connections and Action Code

You can always add to or modify the action code that XVT-Design generates. Keep in mind that XVT-Design doesn’t interpret or execute any of your application’s action code. XVT-Design maintains connections separately from the code fragments. Hence you can have a connection with no code fragment (and vice versa).

Remember that you can specify the default code that Design generates by editing the **design.cft** file. See Chapter 6 for further details on editing this file.

3.7.2. Message Window Menubar

You will create two connections for the window's menubar, to accomplish the following:

- Open a new window when New is chosen from the File menu (the same as for the task window's menubar)
- Bring up the Other Choices dialog when From Dialog is chosen from the Choices menu

Tip: To create a connection for a window's menubar,

1. For the first connection, set the context in the Action Code Editor to:

WIN_MENUBAR	ITEM:New	EVNT:Select

2. Click Connections, and set the connection in the dialog just as you did for the task window (see instructions on the previous page).

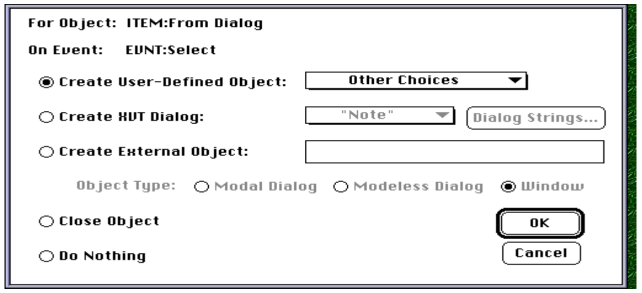
Notice that even though you have only created one window resource, at runtime the application can create any number of windows using this resource. You don't have to create a separate window resource for every window the application might create. In fact, application users can create as many windows as they like. The windows initially will have the same size and location, but can be moved independently and can display different messages.

3. For the second connection, set the context in the Action Code Editor to:

WIN_MENUBAR	ITEM:From Dialog...	EVNT:Select

4. Open the Connections dialog, click Create User-defined Object, and choose Other Choices from the list button.

After you've completed these operations, the Connections dialog looks like Figure 3.26.


For Object: ITEM:From Dialog

On Event: EVNT:Select

[✓ Create User-Defined Object:] Other Choices

Create XVT Dialog: "Note" Dialog Strings...

Create External Object:

Object Type: Modal Dialog Modeless Dialog [✓ Window]

Close Object

Do Nothing

Figure 3.26.Connecting the “Other Choices” dialog to the Choices menu

5. Click OK to dismiss the Connections dialog.

3.7.3. Message Window Connections

The application’s window has one button, which will serve to demonstrate the pre-defined dialogs available in XVT-Design. (You can pretend that this button represents a feature in the application that has not yet been implemented.)

6. Set the context in the ACE to:

Message	PB:Custom String	EVNT:Control

This context tag corresponds to the event of the user clicking the button in the application’s window.

7. Click Connections, and check Create XVT Dialog in the Connections dialog.
8. Set the list button to “Note,” and click Dialog Strings.
9. In the small dialog that opens, enter “Not Yet Implemented!” in the dialog’s edit control.

   This is the message that will be displayed when the button in the application’s window is clicked.

10. Click OK to dismiss the string dialog, then click OK in the Connections dialog to dismiss it.

3.7.4. Other Choices Dialog Connections

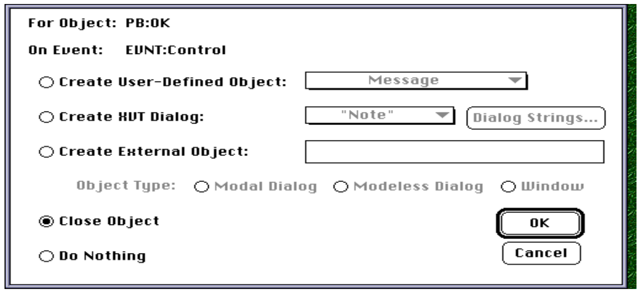
The connections for the Other Choices dialog are quite simple: when the user clicks OK or Cancel, the dialog should go away. For both buttons, you will create a connection that closes the dialog. (You must also handle the dialog's radio buttons, but that comes later.)

Tip: To make the connection for the OK button

11. Set the context to:

Other Choices	PB:OK	EVNT:Control

12. Click Connections to open the Connections dialog.
13. Click Close Object, then click OK to dismiss the dialog.



For Object: PB:OK
On Event: EVNT:Control

( ) Create User-Defined Object: Message
( ) Create XVT Dialog: "Note"
( ) Create External Object:
Object Type: ( ) Modal Dialog ( ) Modeless Dialog ( ) Window
(●) Close Object
( ) Do Nothing

Figure 3.27.Creating a "Close Object" connection

Now, when the OK button in the Other Choices dialog is clicked, its connection closes the dialog.

14. Set the same connection for the Cancel button, using this context:

Other Choices	PB:Cancel	EVNT:Control

3.7.5. About Hello Dialog Connection

The connection for the button in the About box is the same as those in the Other Choices dialog. When the user clicks the button, the dialog should disappear.

Tip: To make the connection,

15. First set the context to:

About Hello	PB:OK	EVNT:Control

16. Then set the connection to Close Object, and click OK.

3.8. Running TestMode

So far, you have created all of the user-interface objects for the sample application, and have defined connections between them. Now you'll use XVT-Design's TestMode to check your work. With TestMode, you verify the appearance of the windows, dialogs, and menus, *without* compiling, linking, and running the application.

Testing the Hello Application

Tip: To test the Hello application,

1. Choose Begin TestMode from the Tools menu.

   XVT-Design hides any open layout and Action Code Editor windows, and replaces its menubar with your application's task window menubar.

2. Answer "Save" to the dialog box that appears in order to your project.

Testing Connections

Tip: Now test some of the connections you created.

3. Choose New from the File menu, and the Message window appears (as shown on the next page).

   Notice that it has the correct menubar—the one you named WIN_MENUBAR and associated with the Message window resource.

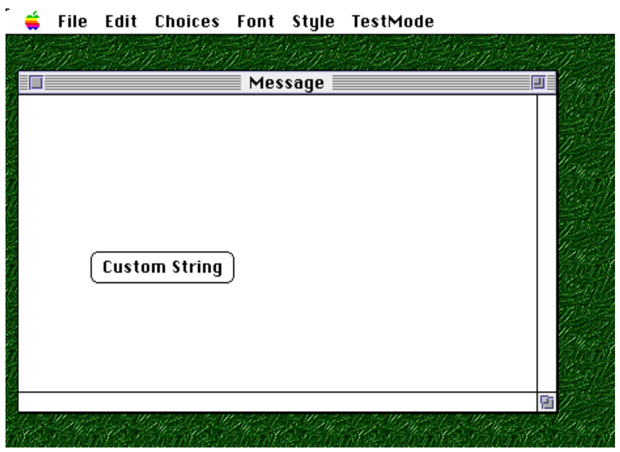


Figure 3.28.Message window showing a menubar and button

4. Try moving and resizing the window—it behaves as you would expect a GUI document window to behave.
5. If you click Custom String, a dialog with the sample error message (“Not Yet Implemented”) appears.

   Recall that you didn’t have to define the dialog explicitly—you asked XVT-Design to use a pre-defined dialog, and gave it the string to display. Click OK to dismiss the dialog.
6. Try opening the Other Choices dialog, by choosing From Dialog from the Choices menu.

   At this stage, the radio buttons won’t do anything if you click them, but the OK and Cancel buttons do dismiss the dialog, as intended.

   Close the Other Choices dialog box.
7. Finally, choose End TestMode from the TestMode menu.

   XVT-Design added this menu to the application’s menubars to provide a way to leave TestMode. This menu is added only in TestMode. It will not appear in the final, compiled application.

   You can also exit TestMode by, from the File menu, choosing Exit or Quit.

Making Adjustments to Interface Objects

You may have noticed that some of the user-interface objects needed minor adjustments. Perhaps you didn't like the location of the Messages window, or maybe the buttons in the Other Choices dialog were not arranged correctly.

You can fix these problems by adjusting the location of the resources, and re-entering TestMode to check your work.

XVT-Design lets you refine the application's appearance without the time-consuming process of editing and compiling resource and source code files, linking the modules, and running the application.

3.9. Attaching Action Code to Tags

With connections, you have implemented much of the application's user-interface behavior; however, you still need to define a few features in source code. Specifically, you must implement the following features:

- Displaying the appropriate message in the Message window, when the user chooses Hello or Goodbye from the Choices menu
- Placing a check mark next to the Hello or Goodbye menu item, after the user chooses one of them
- Displaying the appropriate message in the Message window, after the user clicks OK in the Other Choices dialog
- Changing the appearance of the message when the user chooses commands from the Font/Style menus

To add these features, you'll use the Action Code Editor to create action code for several tags.

Note: Remember that XVT-Design allows you to recover code *after the application has been generated*. See Chapter 4 for details.

Tip: You might find it useful to refer to the *XVT Portability Toolkit Guide* as you complete this part of the tutorial.

3.9.1. Storing the Message

The application's windows will display a message, which is held in a string variable. Because the application can have several windows open at once, and since each window can have a different message, you need to associate a string variable with each window.

The window's application data pointer will hold the message string. When a window is created, you allocate some memory for the string, put a default message into the string, and set the window's application data variable to point at the string.

When a window is created, you also initialize its menubar by setting the Font menu to reflect the default font.

1. Set the context of the Action Code Editor like this:

Message	WIN:Message	EVNT:Create

2. and enter the following code into the ACE's editing pane:

```
XVT_FNTID theFont;
/*
        Allocate space for a text string, and set the
        window's app data field to that pointer. Set the
        initial string to "Hello," which will be displayed
        when the update event for this window occurs.
*/
char *s;
if ((s = xvt_mem_alloc (100)) == NULL)
   xvt_dm_post_error("Insufficient memory");
else {
   strcpy(s, "Hello!");
   xvt_vobj_set_data(xdWindow, PTR_LONG(s));
}
/*
        Initialize the Font menu.
*/

theFont = xvt_dwin_get_font(xdWindow);
xvt_menu_set_font_sel(xdWindow, theFont);
```

Tip: This is about the largest block of code you should ever enter into the Action Code Editor. Generally, XVT recommends that you place a call to a function instead, and define the function in an external file. You then instruct the makefile to include the external file containing the function as part of the project's compilation.

Since memory is allocated when you create the window, you must free it when the window is destroyed.

3. Set the context of the ACE like this:

Message	WIN:Message	EVNT:Destroy

4. and enter the following code:

```
/*
   Free the string that's attached to the window's
        app data.
*/
char *s = (char*)xvt_vobj_get_data(xdWindow);
if (s != NULL)
   xvt_mem_free(s);
```

3.9.2. Displaying the Message

When a window receives an update event, its contents need to be redrawn. In this application, you will erase the window and draw the string you placed there previously.

Tip: To add code to handle the update event,

5. Set the ACE's context to:

Message	WIN:Message	EVNT:Update

XVT-Design has already added default code to erase the window's contents. Your code goes immediately after the default code. The code that XVT-Design generated is shown in italics:

```
/*
   Clear the window to the default background color,
        and draw the text string that's referenced in
        the window's app data field.
*/
xvt_dwin_clear(xdWindow, (COLOR)xvt_vobj_get_attr
        (xdWindow, ATTR_BACK_COLOR));
xvt_dwin_draw_text(xdWindow, 20, 50,
        (char *)xvt_vobj_get_data (xdWindow), -1);
```

3.9.3. Changing the Message with Menu Commands

When the user chooses Hello or Goodbye from the Choices menu, the corresponding string should appear in the Message window. For the Select tag of both of these menu items, you'll add action code that copies a string into the window's message string. Also, you'll force an update event for the window to occur, so that the new message will be redrawn.

Tip: For the Hello item,

6. Set the context like this:

WIN_MENUBAR	ITEM:Hello	EVNT:Select

7. and enter the following code:

```
/*
   Copy the new string into the buffer, and force an
        update. Also, check the appropriate menu item in
        the Choices menu.
*/
strcpy((char *)xvt_vobj_get_data(xdWindow),
        "Hello!");
xvt_dwin_invalidate_rect(xdWindow, NULL);
menu_fix(xdWindow, xdEvent);
```

(The last line of code is a call to a function you haven't written yet. You will write it shortly.)

The action code for the Goodbye item is almost identical:

WIN_MENUBAR	ITEM:Goodbye	EVNT:Select

```
/*
   Copy the new string into the buffer, and force an
        update. Also, check the appropriate menu item in
        the Choices menu.
*/
strcpy((char *)xvt_vobj_get_data(xdWindow),
        "Goodbye!");
xvt_dwin_invalidate_rect(xdWindow, NULL);
menu_fix(xdWindow, xdEvent);
```

Note: It is sometimes easier to cut and paste functions that are similar instead of typing the code in each time.

3.9.4. Checking the Menu Items

Now you'll enter code to produce the following behavior: when the user chooses Hello or Goodbye from the Choices menu, a check mark appears next to the menu item. When the user chooses From Dialog, the check mark disappears.

You'll call this check mark function from the action code for the Select tag for each of the items on the Choices menu. You'll place it in the menubar's Object Declaration action code.

The Object Declaration Tag (Obj_Decl)

The Object Declaration is a convenient tag for functions that are called from several of a module's action code fragments, but are not called from outside of the module.

8. Set the context like this:

WIN_MENUBAR	MBAR:WIN_MENUBAR	SPCL:Obj_Decl

9. and enter the following code:

```
/*
   Check the appropriate Choices menu item when the
       user selects one of the items.
*/

void
menu_fix(WINDOW win, EVENT *ep)
{
   xvt_menu_set_item_checked(win, M_HELLO,
              ep->v.cmd.tag == M_HELLO);
   xvt_menu_set_item_checked(win, M_GOODBYE,
              ep->v.cmd.tag == M_GOODBYE);
}
```

Notice that this function both places a check mark next to the item the user selected, and removes the check mark from the other item (if it was checked).

3.9.5. Changing the Message with the Choices Dialog

Changing the message based on the user's interaction with the Other Choices dialog is slightly more complicated. The dialog must communicate user actions to the window. Did they press OK or Cancel to close the dialog? Did they click a radio button?

To do this, you'll allocate a temporary string variable, and pass it to the dialog when it is created. The dialog's event handler then handles the string according to the user's actions:

- When the user clicks a radio button, the event handler puts the appropriate message into this string
- If the user clicks Cancel, the event handler clears the string before returning

If the string is empty after the event handler returns, the user clicked Cancel and there is no need to change the message. On the other hand, if the string is not empty, you will copy its contents into the window's message string, and force the contents of the window to be redrawn.

Allocating a Temporary String Variable

10. Set the context like this:

WIN_MENUBAR	ITEM:From Dialog…	EVNT:Select

XVT-Design has already added some code for this tag, because you set a connection for it. You will add code both before and after this existing code, and change the connection code slightly to pass the string variable to the dialog's event handler.

11. Enter the following code (the code previously generated by XVT-Design is shown in italics):

```
/*
        Allocate a temporary string variable, and pass it
        to the dialog's event handler. After the event
        handler returns, copy the string into the window's
        message string and force a redraw of the window.
*/
char *s;
if ((s = xvt_mem_alloc (100)) == NULL)
   xvt_dm_post_error("Insufficient memory");
else {
        strcpy(s, "");
        if (!xvt_dlg_create_res(WD_MODAL, DLG_CHOICES,
                EM_ALL, DLG_CHOICES_eh, PTR_LONG(s)))
                xvt_dm_post_error("Can't open dialog");
        if (strlen(s) > 0)
        {
                strcpy((char *)xvt_vobj_get_data(xdWindow),
                s);
                xvt_dwin_invalidate_rect(xdWindow, NULL);
                menu_fix(xdWindow, xdEvent);

        }
        xvt_mem_free(s);

}
```

Caution: When you enter the above code, be sure to change the last parameter in the xvt_dlg_create_res call to PTR_LONG(s), as shown in bold.

The dialog itself needs code to accomplish the following:

- When any radio button is clicked, the button's title must be copied into the dialog's application data (that is, the string allocated before creating the dialog)
- If OK is clicked, the dialog closes, leaving the button title in the previously allocated string, so that it can be copied to the window's message string
- If the user clicks Cancel, the string is erased so that the message is left unchanged

XVT-Design creates default action code for radio buttons, which handles checking and unchecking the buttons as a group. You'll leave this code in place, and add code that copies the title of the radio button into the dialog's application data.

Adding Code for the Radio Buttons

12. Set the context to:

Other Choices	RB:Have a nice day!	EVNT:Control

13. And add the following code *after* the existing code in the edit pane:

```
/*
        copy the button's title into the dialog's app data
*/
xvt_vobj_get_title(xvt_win_get_ctl(xdWindow,
        DLG_CHOICES_RADIOBUTTON_1), (char *)
        xvt_vobj_get_data(xdWindow), 100);
```

The action code is almost exactly the same for each of the other radio buttons. For each one, set the context as shown, and enter the code that follows it. You might find it convenient to use the Copy and Paste commands on the Edit menu to paste copies of text in each context.

Other Choices	RB:See ya later, alligator!	EVNT:Control

```
/*
        copy the button's title into the dialog's app data
*/
xvt_vobj_get_title(xvt_win_get_ctl(xdWindow,
        DLG_CHOICES_RADIOBUTTON_2), (char *)
        xvt_vobj_get_data(xdWindow), 100);
```

Other Choices	RB:Beam me up, Scotty!	EVNT:Control

```
/*
        copy the button's title into the dialog's app data
*/
xvt_vobj_get_title(xvt_win_get_ctl(xdWindow,
        DLG_CHOICES_RADIOBUTTON_3), (char *)
        xvt_vobj_get_data(xdWindow), 100);
```

Other Choices	RB:Make it so!	EVNT:Control

```
/*
        copy the button's title into the dialog's app data
*/
xvt_vobj_get_title(xvt_win_get_ctl(xdWindow,
        DLG_CHOICES_RADIOBUTTON_4), (char *)
        xvt_vobj_get_data(xdWindow), 100);
```

Adding Code for the Push Buttons

Finally, you must enter code for the push buttons. Both buttons already have action code for their tags, which XVT-Design inserted when you created connections for the buttons.

14. The OK button needs no additional code, but the Cancel button needs code to clear the application data string:

Other Choices	PB:Cancel	EVNT:Control

15. Add this text *before* the existing code:

```
/*
        Clear the application data, so that the string
        will not replace the window message
*/
strcpy((char *) xvt_vobj_get_data(xdWindow), "");
```

3.9.6. Changing the Font and Style

The last action code you must create deals with the Font/Style menus. The XVT Portability Toolkit handles most of the details—you simply need to call the appropriate functions at the right time (that is, when the user chooses an item from the Font/Style menu).

Message	WIN:Message	EVNT:Font

```
/*
        Set the window's font, update the check marks on
        the Font menu, and force the window to redraw its
        contents.
*/
xvt_dwin_set_font(xdWindow,
         xdEvent->v.font.font_id);
xvt_menu_set_font_sel(xdWindow,
        xdEvent->v.font.font_id);
xvt_dwin_invalidate_rect(xdWindow, NULL);
```

3.10. Generating the Application

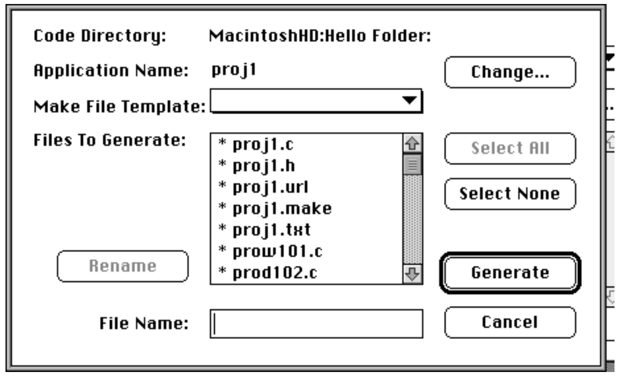
At this point, you have created all of the application's resources, made connections between them, and added action code. Now you will generate the source code files. XVT-Design will create complete a source code, a header, and a resource file, as well as a platform-specific makefile to build the complete application.

3.10.1. Setting the Application Name

Before XVT-Design generates files for the application, you must tell it where to put the files, and provide an application file.

1. Choose Generate Application from the File menu.

The dialog shown in Figure 3.29. opens.



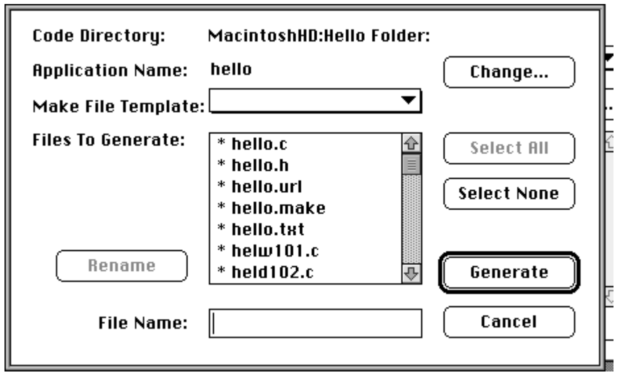
Code Directory: MacintoshHD:Hello Folder:
Application Name: proj1
Make File Template:
Files To Generate:
* proj1.c
* proj1.h
* proj1.url
* proj1.make
* proj1.txt
* prow101.c
* prod102.c
File Name:

Figure 3.29.The Generated Files dialog

2. Click Change, and a standard file save dialog appears.
3. Navigate to the directory in which you want to place the generated code files, which might probably be the same one that contains your project file.
4. In the Name field, enter the name of the finished application: “hello.”
5. Click Save to dismiss the file save dialog.

In the center of the Generate Application dialog, a list box shows the names of the files that XVT-Design will generate for your application.

Notice that they have been changed to reflect the name you just gave to your application, as shown in Figure 3.30.


Code Directory: MacintoshHD:Hello Folder:
Application Name: hello
Make File Template:
Files To Generate:

```
* hello.c
* hello.h
* hello.url
* hello.make
* hello.txt
* helw101.c
* held102.c
```

File Name:

Figure 3.30.New names for the generated files

Selecting a Makefile Template

A list button in the Generate Application dialog, labeled “MakeFile Template:”, contains the makefile templates XVT-Design knows about on your platform.

6. Select the makefile for your compiler if it uses a makefile. Otherwise, double-click on hello.make to stop a makefile from being generated. (The asterisk next to hello.make will disappear.)

3.10.2. Generating the Source Files

To generate source files, click Generate. XVT-Design now creates a source code, a makefile, a header, and a resource file in the directory you specified in the previous step. You may want to examine these files to see how XVT-Design incorporated the action code you created with its structural code.

7. When the dialog tells you that application generation was successful, click OK.

3.11. Building and Running the Application

How you actually build the final application depends on your development environment:

- If you’re using an environment with makefiles, XVT-Design will have created a complete makefile for your application.

Use **make** (or whatever your make utility is called) to compile the application.

- If you're using an environment without makefiles, use the project file in the **tutorial** directory in your XVT-Design installation.

8. Once you've compiled the application, run it and test the features constructed in this tutorial. Open several windows and set a different font and style in each. Try changing the messages, first by choosing Hello and Goodbye from the Choices menu, and then by opening the Other Choices dialog and clicking different radio buttons.

3.12. XVT-Design and Beyond

This tutorial has demonstrated how XVT-Design can speed the creation and layout of application resources such as windows and dialogs, controls, menus and strings.

You have seen how to create a project, create and lay out GUI objects, set application attributes, add and test connections, add action code, and generate an application. Now that you have finished this tutorial, you can learn more about XVT's Development Solution for C:

- See the *XVT Portability Toolkit Guide.*

4

USING XVT-DESIGN

This chapter describes the main features of XVT-Design, and tells how to use them to build your application. The chapter covers the following topics:

- Project Files
- Using the Action Code Editor (ACE)
- Creating Windows, Dialogs, and Controls
- Layout Commands
- Object Attributes
- Creation Order
- The Menu Editor
- String Resources
- Userdata Strings
- Help with Help Files
- TestMode
- Generating Source Code
- Code Recovery

4.1. Project Files

For every application you build with XVT-Design, you begin by creating a new project file. Project files are the “documents” you create and modify with XVT-Design. A project file contains all the resources and source code for your application’s user interface.

Projects are stored in binary files. These files are portable across all platforms on which XVT-Design runs. You can create a project file on one platform, then move it to another platform—without modification—to develop and refine your application on both platforms.

4.1.1. Creating New Projects

Tip: To create a new project:

Choose New Project from the File menu.

XVT-Design creates a new project and opens an Action Code Editor window and a layout window for the project.

4.1.2. Project Attributes

There are several attributes which affect your application as a whole. You set these attributes with the Project Attributes dialog.

Tip: To open this dialog:

Choose Project Attributes from the Edit menu.

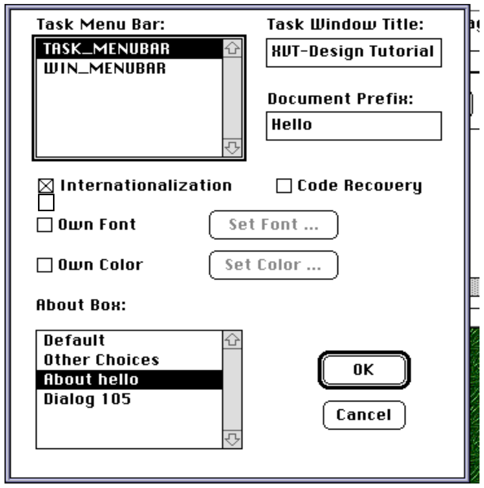

Task Menu Bar:
TASK_MENUBAR
WIN_MENUBAR
Task Window Title: XVT-Design Tutorial
Document Prefix: Hello
☒ Internationalization
☐ Code Recovery
☐ Own Font
☐ Own Color
About Box:
Default
Other Choices
About hello
Dialog 105

Figure 4.1.The Project Attributes dialog (Macintosh Platform)

You can set the following attributes with this dialog:

Task Menubar

The task menubar is the menubar shown in the task window. The Project Attributes dialog has a list box that contains the names of all of the menubars in the project. Click the appropriate name to select the task window's menubar.

All new project files contain a default menubar, named TASK_MENUBAR, which includes the standard File, Edit, Font and Help menus. This menubar appears first in the list box.

Task Window Title

The title of your application's task window is usually the same as the name of your application. To set the task window's title, type its name in the edit control.

Document Prefix

The document prefix string is placed at the beginning of the names of new document windows in your application. To set the document window name prefix, type it in the edit control.

Internationalization

You can easily generate internationalized applications with XVT-Design. The program supports efficient localization process and allows flexibility in internationalization schemes

See Also: For details on the entire internationalization process, see Chapter 6 of this manual.

Code Recovery

With XVT-Design you can edit the generated files using any text editor and later recover code from inside of XVT-Design.

Code fragments are wrapped with special comments during the code generation process. You can edit the files using a standard text editor and recover all the changes you have made to the generated files.

To make the most of this feature, it must be activated by clicking on the Code Recovery radio button (in the Project Attributes dialog box) early in the design process.

See Also: For further explanation of code recovery, see *Code Recovery* on page 4-68.

Own Font

You can set the default control font for the entire application by clicking on this button.

See Also: For further explanation of choosing and setting fonts, see *Fonts and Colors* on page 4-26.

Own Color

You can set the default control color for the entire application by clicking on this button.

See Also: For further explanation of choosing colors, see *Fonts and Colors* on page 4-26.

About Box

The About Box is a dialog that is invoked when the user of your application chooses the “About…” menu item. You can either use a default dialog supplied by XVT-Design, or create your own. Any modal dialog in your project can be used.

The Project Attributes dialog has a list box that contains the names of all of the dialogs in the project. Click the appropriate name to select the About box.

4.1.3. Working with Multiple Projects

You can work with more than one project at a time. The names of all open projects are listed at the bottom of the Edit menu.

Only one project is *active* at a given time. The active project has a check mark next to its name on the Edit menu. The windows for any inactive projects are hidden. To make a project active, choose its name from the Edit menu.

4.2. Using the Action Code Editor (ACE)

In XVT-Design, the Action Code Editor (ACE) is the primary tool for creating and editing your application’s user-interface source code. Using the ACE, you can

- Create and edit user action code
- Create and modify connections for TestMode
- Invoke other XVT-Design editors

This section describes how to use the ACE for each of these functions.

4.2.1. Invoking the ACE

Tip: You can invoke the ACE in several ways:

- From the Tools menu, choose Action Code Editor. You can use this method at any time, as long as a project is open.
- From the Edit menu, choose Edit Code. This menu item is available only when the active window is a layout window for a window or dialog.
- In a layout window, hold down the Shift key and double-click on a control, or the layout window itself.

4.2.2. ACE Code Fragment Templates

In previous version of XVT-Design, the initial ACE code for each tag was hardwired. In this new version of Design, initial ACE code is contained in a text file called **design.cft**.

- The **design.cft** file can be customized.
- When XVT-Design starts up, it reads **design.cft**.
- *Each tag whose code fragment has not previously been modified in the ACE gets initialized with the code you have specified.*
- You can create multiple versions of **design.cft** for use with different applications.

Note: It is important to note that ACE code fragments should *not* be created larger than 32K.

4.2.2.1. Editing the Design.cft file

The **design.cft** file is located in the same directory as **design.cfg**.

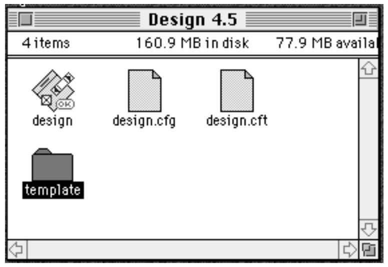
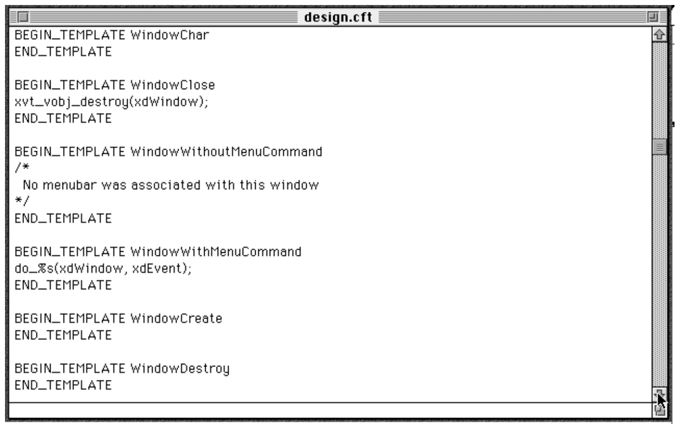


Figure 4.2.Directory for the design.cft file

4.2.2.2. Viewing the design.cft file

You can use any text editor to open the file.

The **design.cft** file is shown in the diagram below. There are two types of string variables and they are detailed in the file itself.



```
BEGIN_TEMPLATE WindowChar
END_TEMPLATE

BEGIN_TEMPLATE WindowClose
xvt_vobj_destroy(xdWindow);
END_TEMPLATE

BEGIN_TEMPLATE WindowWithoutMenuCommand
/*
 No menubar was associated with this window
*/
END_TEMPLATE

BEGIN_TEMPLATE WindowWithMenuCommand
do_%s(xdWindow, xdEvent);
END_TEMPLATE

BEGIN_TEMPLATE WindowCreate
END_TEMPLATE

BEGIN_TEMPLATE WindowDestroy
END_TEMPLATE
```

Figure 4.3.A portion of the design.cft file

4.2.2.3. Rules for Editing design.cft

- Do not add or remove any templates.
- Do not change the BeginTemplate or EndTemplate statements.
- Do not add or remove %'s – XVT-Design expects a *specific number* of %'s for each template.
- If you change the design.cft file after a project has been created, note that only tags which have never been modified are updated from the design.cft file.

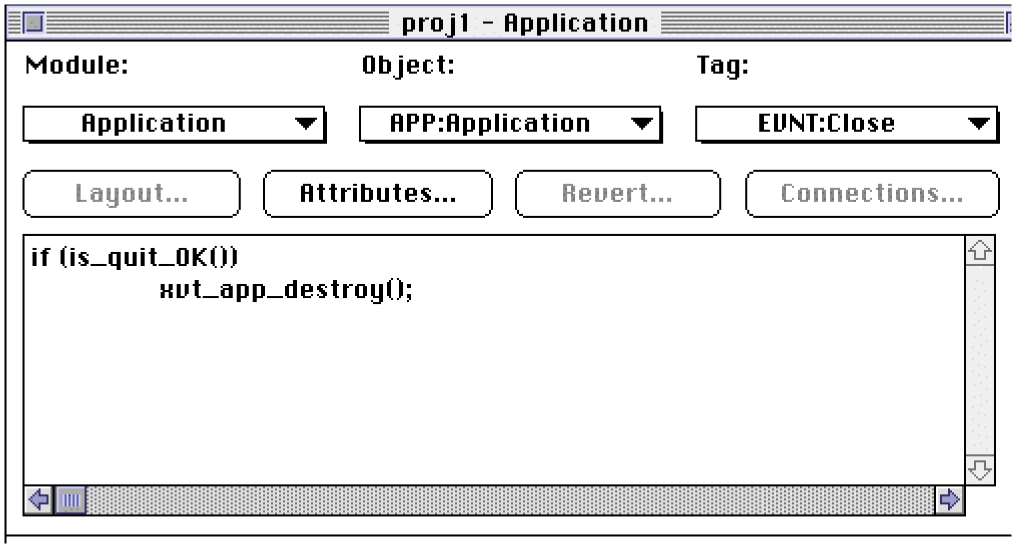
4.2.2.4. Resolving %s's.

%s's are documented in the **design.cft** file itself. There are two types of %s's:

- *A string variable that references a variable that XVT-Design knows about*. These variables are like printf functions in C. Do **not** change these types of variables**.**
- *A string variable that references internal values*.
  This kind of variable **can** be changed.

4.2.3. ACE Controls

When opened in a new project, the ACE window looks like this:



```
if (is_quit_OK())
        xvt_app_destroy();
```

Figure 4.4.Controls in the Action Code Editor (Macintosh Platform)

The ACE window contains several controls: three list buttons for setting the editing context, a text editing region for examining and modifying source code, and several push buttons for other operations. The following section describes the function of each control.

4.2.3.1. The Editing Context

In the ACE, *context* refers to a specific tag for a specific object. The context consists of three parts: the module that contains the object, the object itself, and the tag. Three list buttons in the ACE correspond to these parts (see below).

The titles of the list buttons always indicate the current context. Action code can be associated with each context. The action code is displayed in the text editing pane of the ACE.

Module

The Module list button lists the titles of all the containers and menubars in the current project. An additional item, “Application,” refers to the context for application events. The code for each module in the project is generated into a different C file.

Object

The Object list button lists the titles of all the objects contained by the item specified by the Module list button:

- If the Module item is a window or dialog, the objects are the controls in that window or dialog
- If the Module item is a menubar, the objects are the titles of the menu items
- If the Module item is "Application," there is only one object, also titled "Application"

Items in the Object list button have a prefix that indicates their type:

Containers	
DLG: Dialog	WIN: Window

Controls	
CB: Check Box	LX: List Box
CC: Custom Control	PB: Push Button
ED: Edit Control	RB: Radio Button
HS: Horizontal Scrollbar	TE: Text Edit
LB: List Button	TX: Static Text
LE: List Edit	VS: Vertical Scrollbar

Tag

The Tag list button lists all the tags available for the item indicated by the Object list button. The items on this list vary depending on the kind of object (control, menu item, etc.) in the context. There are two types of tags:

- *Event* tags have the prefix "EVNT:"
- *Special* tags have the prefix "SPCL:"

4.2.3.2. The Text Editing Pane

The center of the ACE window contains a rectangular pane for editing text. You'll use this editor to create, modify, and examine all the action code fragments for your application. The code in this pane always corresponds to the current context of the ACE, as shown by the three list buttons.

If the code won't fit in the editing pane, you can use the horizontal and vertical scroll bars to view it. You can change the size of the pane by resizing the ACE's window.

To edit text in the ACE, you can use three Edit menu commands:

Cut
Removes the selected text from the editor and places it on the system clipboard.

Copy
Places a duplicate of the selected text onto the system clipboard.

Paste
Places the text from the system clipboard in the editor, at the insertion point. If text is selected in the editor, the clipboard text replaces it.

All text created with the editor is stored on disk in your project file when it is saved. When XVT-Design generates the source files for your application, the text is copied into the appropriate files.

You can also recover code after it has been generated.

See Also: For further explanation of code recovery, see *Code Recovery* on page 4-68.

See Also: You can change the font and font size used in the ACE text pane by editing XVT-Design's configuration file. See *Fonts and Colors* on page 4-26.

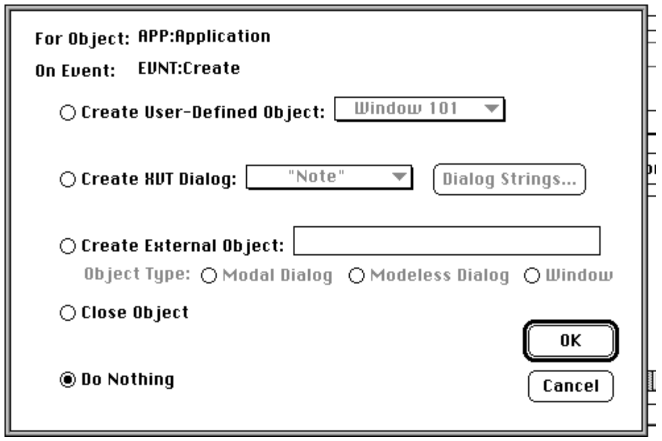
4.2.3.3. Creating and Editing Connections

You create connections in the ACE, using the Connections button. Connections provide a link between user interface objects in your application. For example, you can make a connections between a push button and a window; once the connection is made, when the button is pushed, the window appears.

Adding a connection usually means that XVT-Design generates code and places it in the ACE editing window.

Connections

Brings up the Connection editing dialog (see following illustration). This button is enabled only when the context is set to an object and tag that allows a connection. In other words, it is enabled only when the context is a menu item's Select event, a push button's Control event, or the application's Create event.


For Object: APP:Application

On Event: EUNT:Create

○ Create User-Defined Object: Window 101

○ Create XUT Dialog: "Note" Dialog Strings...

○ Create External Object:

Object Type: ○ Modal Dialog ○ Modeless Dialog ○ Window

○ Close Object

[✓ Do Nothing]

Figure 4.5.The Connection Dialog (Macintosh Platform)

The Connection dialog displays the object title and tag near the top to indicate the context for the connection. Radio buttons on the left indicate the action that will be executed when the event occurs:

Create User-defined Object

If this box is checked, the connection creates one of your application's windows or dialogs. A list button, enabled only when this radio button is checked, indicates which one will be created. From the list, choose the name of the object that the connection will create.

Create XVT Dialog

If this box is checked, the connection invokes one of the pre-defined XVT dialogs. A list button, enabled only when this radio button is checked, shows the names of the pre-defined XVT dialogs:

Question
A dialog with a question icon, a message, and two or three buttons, used to query the application user.

Error
A dialog with an error icon, a message, and an OK button.

Note
A dialog with a note icon, a message, and an OK button.

Message
A dialog with an information icon and a message, useful in low-memory or error conditions.

Save
A native save-file dialog, which allows the application user to enter a name for a file and to choose its directory.

Open
A native open-file dialog, which allows the application user to open a file.

Font
A dialog which allows the application user to choose a font and type style.

From the list, choose the name of the dialog that the connection will invoke.

Dialog Strings

To enter the message that the dialog displays, click the Dialog Strings button. A small dialog opens; enter the message in the dialog's edit control and click the OK button.

If the connection creates the XVT "Question" dialog, you can also enter titles for the three buttons in the dialog.

Create External Object

If this button is checked, XVT-Design creates an external connection—that is, a connection to an object outside of the current project. You must supply the symbolic identifier for the object (for example, WINDOW_101), and click the appropriate radio button to indicate the type of the object.

Tip: To create a connection to an external object:

1. Click the Create External Object radio button.
2. Type the symbolic identifier of the object in the edit field.
3. Click the radio button that corresponds to the type of the external object—Modal Dialog, Modeless Dialog, Modal Window or Window.

Close Object

If this button is checked, the action closes the window or dialog that contains the object. Typically this connection is used to dismiss a dialog or close a window when a button in that container is pushed.

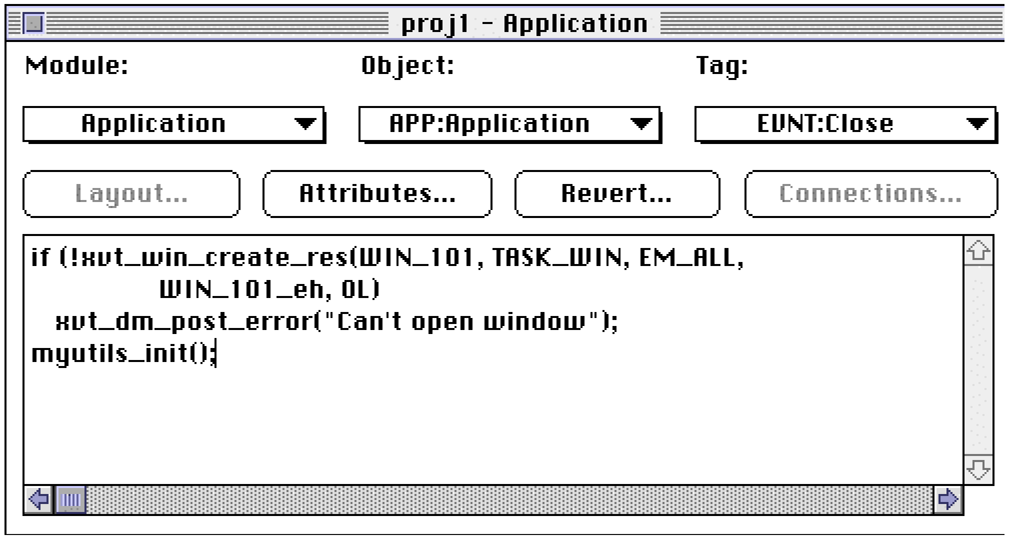
Do Nothing

If this button is checked, the connection will not have any effect, and no code will be generated for it. This button is checked by default when the connection dialog is first opened for a context. To remove an existing connection, check this button.

Note: Only the internal connection that XVT-Design uses for TestMode is removed. You must remove the connection's action code from the ACE by hand.

4.2.3.4. Using Other ACE Controls

The ACE includes four other push buttons for performing various operations:



```
if (!xvt_win_create_res(WIN_101, TASK_WIN, EM_ALL,
          WIN_101_eh, 0L)
  xvt_dm_post_error("Can't open window");
myutils_init();
```

Figure 4.6.Action Control Editor

Layout

Brings up the layout window for the context object. If the context is a menubar, the menubar editor opens. (See section 4.7.2 on page (4-50).)

Attributes

Opens the Attributes dialog for the context object.

Revert

May perform one of the three following actions (depending on what code is in the Text Box):

Revert to Previous

- Discards any changes you have made in the text editing pane. The pane reverts to the text that was present when the context was last selected.

Revert to Default Code

- Reverts to default code. The pane reverts to the default code (replaces the current content of ACE with default code).

Append Default Code

- Appends default code. XVT-Design supplies action code for some object/tag combinations. This “generic” code suggests what your application should do by default for the context. This code appears in the text editing pane; you can edit it just as you would edit code you create yourself. After editing the code, you can restore the original generic code by clicking the Default Code button. The default code is added at the end of any existing text.

Connections

When you create a connection, XVT records the connection in the project file. The action code is also added to the module’s source code so that the connection will occur when the application is generated.

4.2.3.5. Finding Text in Action Code

The Edit | Find command brings up a dialog box that allows you to set options that control a search through action code for the first match of a string that you specify. The search applies to the active Action Code Editor (ACE).

When you click the Find button in the dialog, the Action Code Editor is positioned to the tag containing the match, and the matching text is selected. You may then edit the text, or perform any other XVT-Design operations.

Use the Edit | Find Next command to find the next match; if you've moved to another tag, it finds the next match after that tag. In other words, it starts the search from the current position. Or, you can issue the Edit | Find command again, using the same or different options in the dialog box.

With both the Find and Find Next commands, the search doesn't go beyond the scope (see below). If no match is found, the position of the ACE is left undisturbed.

These are the controls in the Find dialog:

Search for

The text to be searched for.

Case Sensitive

If checked, the match is case sensitive; otherwise, it is case insensitive.

Whole Words Only

If checked, the text in the "Search for" field must match complete words. (That is, "break" will not match "breakfast".)

4.2.3.6. Controls in the Origin Groupbox

From Cursor

If selected, the match begins from the current caret position in the current action code.

Entire Scope

If selected, the match begins at the start of the scope, which is set by one of the following four radio buttons.

4.2.3.7. Controls in the Scope Groupbox

Project

If selected, the scope is the entire project. If Entire Scope is selected, the search begins with the first tag for the Application module and the "APP:Application" object; otherwise it begins with the current caret position. In both cases it ends with the last tag for the last object for the last module.

Module

If selected, the scope is the entire module. If Entire Scope is selected, the search begins with the first tag for the first object in the module; otherwise it begins with the current caret position. In both cases it ends with the last tag for the last object in the module.

Object

If selected, the scope is the entire object. If Entire Scope is selected, the search begins with the first tag for the object; otherwise it begins with the current caret position. In both cases it ends with the last tag for the object.

Tag

If selected, the scope is the entire action code for the tag. If Entire Scope is selected, the search begins at the start of the action code; otherwise it begins with the current caret position. In both cases it ends at the end of the action code.

Find

When this button is pressed, the dialog is dismissed and the search begins.

Cancel

This button dismisses the dialog without starting a search.

4.2.3.8. Scanning a Project's Action Code

The Edit | Scan Tags command allows you to quickly review the action code for part or all of a project.

4.3. Creating Windows, Dialogs, and Controls

XVT-Design provides tools for creating windows, dialogs, and controls graphically and interactively. You create and adjust these resources directly on your screen, and XVT-Design generates the appropriate resource description files.

See Also: For more information on the differences between windows and dialogs, see *Project Files* on page 4-1.

4.3.1. Creating Windows and Dialogs

Tip: To create a new window:

1. Choose New Window from the Window menu. XVT-Design opens a new layout window.
2. Move and resize the window to suit your needs. The size and location of the layout window represent the size and location of the window resource you have created.

Tip: To create a new dialog:

1. Choose New Dialog from the Window menu. XVT-Design opens a new layout window.
2. Move and resize the dialog to suit your needs. The size and location of the layout window represent the size and location of the dialog resource you have created.

Note: XVT-Design uses document-style layout windows to represent windows and dialogs of all types. This lets you easily adjust the position and size of these resources. In TestMode, and in your compiled application, the windows and dialogs are rendered appropriately for their type.

4.3.1.1. Modal Windows

XVT-Design supports the layout and generation of modal windows. There is a Window Attributes radio button for Modal Window type. And you can preview these windows in Test Mode.

The purpose of a modal window is to block the users' interaction with any other application window except the modal window itself.

Modal windows have a different look-and-feel on each platform, because they conform with the required style of that platform's window manager.

A modal window prevents user interaction with any other window of an application (including the parent window which may be modal itself) until some user-initiated action causes the modal window to be dismissed. When a user initiates a request for dismissal, the application *must* destroy the modal window by calling xvt_vobj_destroy. After a modal window is destroyed, focus returns to the window which previously had focus.

Notes About Using Modal Windows

- Modal windows do not support menu bars.

- When laying out a modal window, you must have a button inside the window with a connection to *close the object*.

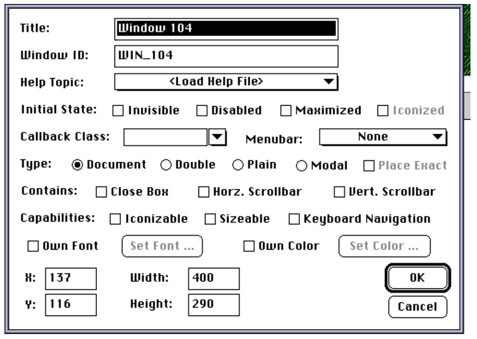
Why use Modal Windows?

Modal windows use several types of objects that are not available in dialogs. Specifically, modal windows can contain custom controls, text edits and child windows.

In addition, modal windows allow drawing operations (while dialogs do not).

Tip: To create a modal window

1. Create a window.
2. Click on its Attributes button in the ACE or, in the Layout Editor, double-click in the background of the window.



Title: Window 104
Window ID: WIN_104
Help Topic: <Load Help File>
Initial State: ☐ Invisible ☐ Disabled ☐ Maximized ☐ Iconized
Callback Class: Menubar: None
Type: [✓ Document] ○ Double ○ Plain ○ Modal ☐ Place Exact
Contains: ☐ Close Box ☐ Horz. Scrollbar ☐ Vert. Scrollbar
Capabilities: ☐ Iconizable ☐ Sizeable ☐ Keyboard Navigation
☐ Own Font ☐ Own Color
X: 137 Width: 400
Y: 116 Height: 290

Figure 4.7.Window Attributes dialog box

3. Click on the Modal radio button.

Modal Window Look-and-Feel

Modal windows are implemented using the native object best suited to providing modality on each platform. A W_MODAL window may have characteristics of a top-level window, a child window, or a dialog. Moreover, the look-and-feel of this object is platform-specific—it will have the physical appearance most appropriate for modality on a particular platform. Modal windows follow native look-and-feel guidelines for decorations (borders, system menus, etc.) and stacking order.

See Also: For further information about Modal Windows, see the *XVT Portability Toolkit Guide*, chapter 6.

4.3.2. Creating Controls

Tip: To create one or more controls:

1. From the Controls menu, choose the control type.
   (The custom control menu item has a hierarchical menu that lists all of the installed custom controls.)
2. Position the cursor in the upper left corner of the desired location.
3. Either click or drag the control into the desired size.
   If you click to create the control, it will be of the standard size for this type of control.
4. Click and/or drag to create additional controls of this type.
5. To exit from this mode, choose Pointer (or another control) from the Controls menu.

Tip: To move a control:

Click and drag it with the pointer.

Tip: To change the size of a control:

1. Click the control once to select it.
2. Drag the small black rectangle near the lower-right corner of the control.

4.3.3. The Object Palette

Layout windows contain a palette of graphical toggle buttons which represent the various controls you can put in a window or dialog.

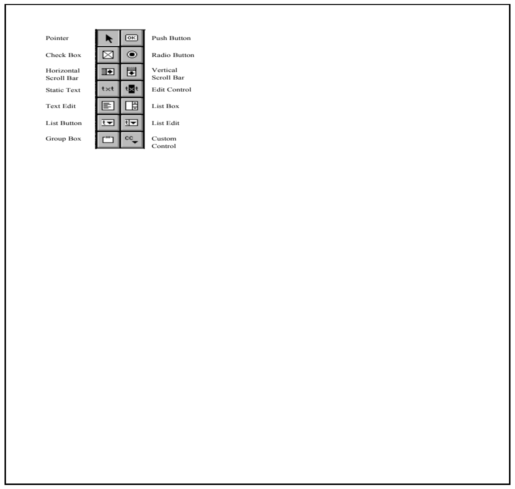


Figure 4.8.The Object Palette

You can use these toggle buttons to create controls, instead of using the commands on the Controls menu. Clicking once on a toggle button in the palette is the same as choosing the corresponding control from the Controls menu. Once you click the toggle button, it remains active until you click a different button. This allows you to create a number of controls of the same type after clicking the toggle button once.

Tip: To create a control using the object palette:

1. Click the button of the desired control in the object palette.
2. Position the cursor in the upper left corner of the desired location.
3. Either click or drag the control into the desired size.
   If you click to create the control, it will be of the standard size for this type of control.

The custom control toggle button behaves slightly differently than the other buttons.

Tip: To create a custom control using the object palette:

1. Click the custom control button in the object palette.
   When you click the custom control button, a menu containing

the names of all of the currently installed custom controls drops down next to the button.

2. Click the appropriate name on the menu to choose a custom control.
   Once you have chosen the control from the drop-down menu, you can create one or more of these controls just as you would create standard controls.

4.3.3.1. Hiding the Object Palette

By default, all layout windows contain an object palette when first opened. You can hide the object palette if you desire. For example, you may want to hide the palette after you have created all of the controls in the container, since you no longer need the palette.

Tip: To hide the object palette:

Choose Hide Object Palette from the Layout menu.

Tip: To show a hidden palette:

Choose Show Object Palette from the Layout menu.

The object palette is hidden on a per-window basis; i.e. you can show the palette in one window and hide it in another.

Note: The presence or absence of the object palette does not affect your container’s appearance at runtime—it is part of the layout window, not of your container itself.

4.4. Layout Windows

A layout window in XVT-Design represents the actual window or dialog in your generated application.

The Layout menu includes various alignment and spacing commands to help you position controls in dialog boxes and windows. When you first create controls, you can position and/or size them manually by using the mouse or entering position and/or size values. Then you can fine-tune them using the layout commands.

4.4.1. Alignment

All alignment operations use the position of the first selected control as a reference point for lining up the other controls. For example, if you select push buttons 3, 2, and 1 (in that order), and then select Align Left from the Layout menu, push buttons 2 and 1 line up along the left border of push button 3, because it was the first control selected.

Align Left

Aligns the selected controls along their left border. The position of the first control selected is the reference point for lining up the other controls.

Align Center

Aligns the selected controls along their horizontal center by moving them left or right.

Align Right

Aligns the selected controls along their right border.

Align Top

Aligns the selected controls along their top border.

Align Middle

Aligns the selected controls along their vertical middle by moving them up or down.

Align Bottom

Aligns the selected controls along their bottom border.

4.4.2. Spacing

Even Horizontal Spacing

Equalizes the horizontal spacing between the right-most and left-most boundaries of the selected controls. If the controls would overlap, XVT-Design places them adjacent to each other horizontally, which increases the total distance between the right-most and left-most boundaries of the selected controls.

Even Vertical Spacing

Equalizes the vertical spacing between the top-most and bottom-most boundaries of the selected controls. If the controls would overlap, XVT-Design places them adjacent to each other vertically, which increases the total distance between the top-most and bottom-most boundaries of the selected controls.

Make Same Size

Makes all selected controls the same size as the first control selected. Controls with the Standard Size attribute set may not be affected by this command, or may be affected in only one dimension.

4.4.3. Grid

The Grid command lets you superimpose a grid on a selected window or dialog box, to help you position controls. You can determine the spacing of the grid, and choose whether controls snap to it.

Grid Spacing

Sets the horizontal and vertical spacing of the grid in pixels (default setting: 8 x 8) or in characters. You can use any size pixel-based grid, depending on how you like to lay out your controls. Or, for greater portability, you can use a character-based grid. (See "Tip," later in this section.)

Snap To Grid

Determines whether controls you create, move, or resize are automatically aligned to the layout grid. As you move a control, its position will jump to the nearest grid line intersection. As you change the size of a control, its right and bottom edges will jump to the nearest grid lines. This option is independent of the Display Grid option; controls can snap to a grid even if it is not displayed.

Note: Controls that you have already placed in the layout window are *not* affected when you select this command, until you move or resize them.

Display Grid

Determines whether the grid is displayed in the selected window or dialog box. This option is independent of the Snap To Grid option; you can display the grid without forcing controls to snap to the grid.

Tip: To ensure that your resources look good when you move your application to other platforms, we recommend that you use the character-based grid. When you move your project from one platform to another, this grid changes size, but the grid spacing is always the size of a character of average width and height in the system font (the default font used to draw labels in native controls). If you use a character-based grid, your controls will map cleanly to coordinate systems on other platforms, and thus be properly aligned and spaced.

4.4.4. The Toolbar

Layout windows have a toolbar, with picture buttons corresponding to the most frequently used commands for creating and editing controls. Clicking a button on the toolbar has exactly the same effect as choosing the corresponding command from the menu.

4.4.4.1. Hiding the Toolbar

By default, all layout windows contain a toolbar when first opened. You can hide the toolbar if you desire.

Tip: To hide the toolbar:

1. Choose Hide Toolbar from the Layout menu.

Tip: To show a hidden toolbar:

1. Choose Show Toolbar from the Layout menu.

The toolbar is hidden on a per-window basis; i.e. you can show the toolbar in one window and hide it in another.

Note: The presence or absence of the toolbar does not affect your container's appearance at runtime—it is part of the layout window, not of your container itself.

4.5. Setting Object Attributes

In XVT-Design, the attributes for your application's windows, dialog boxes, and controls are set with dialog boxes.

Tip: To invoke the attributes dialog box for an object:

1. Select the object (dialog box, window, or control).
2. From the Edit menu, choose Attributes.
   -OR-

   Double-click the object.

4.5.1. Common Attributes

The specific attributes that can be set vary depending on the type of object. The attributes that are common to all objects are described in this section. Object-specific attributes are described in the following sections, according to the type of object.

Title

Allows you to specify a particular title or name for an object. Each object's title field is automatically filled in with a system-defined default name.

Symbolic Identifier

Specifies a symbolic name for the resource ID of an object. These names for resource IDs are placed in a header file that is created when you invoke the Generate Application command. This header file is then included in the **.url** resource files and the **.c** source code files that need access to the resources.

A symbolic identifier lets you symbolically refer to a particular resource within your application code. This field is automatically filled in with a system-defined default name.

Caution: The default symbolic identifiers that XVT-Design creates are unique within a project. If you edit the symbolic identifier field, you must be sure that all symbolic identifiers are unique. You cannot have two objects with identical symbolic identifiers within a project.

X and Y coordinate locations

Specifies the position of an object. The X and Y (horizontal and vertical) coordinate values are specified in pixel units and refer to the position of the upper left-hand corner of the object. For controls, these values are set when you move the object in a layout window. For windows and dialogs, these values are set when you move the layout window itself. You can also type them in edit fields in an attributes dialog.

Height and Width fields

Defines height and width values, in pixels, for a particular object. For controls, these values are set when you changes the size of the object in a layout window. For windows and dialogs, these values are set when you change the size of the layout window itself. You can also type them in edit fields in an attributes dialog.

Help Topic

Associates a help topic with the object. A list button shows all of the help topics in the current help text source file. To associate a topic with the object, select its name on the list button.

Note: Once you have written a help text source file for your application, you must load it into XVT-Design before you can associate topics with objects.

Tip: To load a help text source file into XVT-Design:

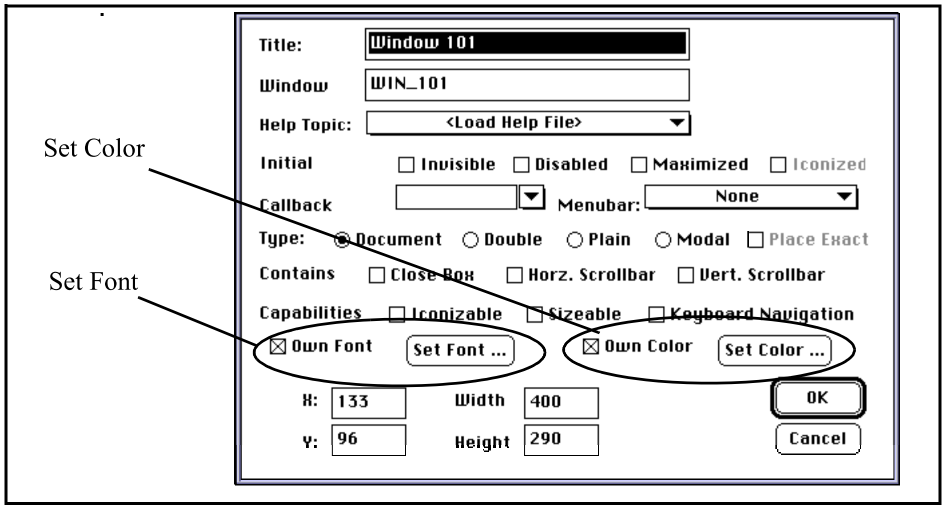
1. Open the attributes dialog for any window, dialog, control, or menu item.
2. Choose the "<Load Help File>" item from the Help Topic list button in the attributes dialog. This invokes a standard open file dialog.
3. Open your help text source file in the open file dialog.

After you open the help text source file, XVT-Design scans the file, extracts all of the help topic titles in the file, and places them in the Help Topic list button shown in the attributes dialogs.

See Also: For more information on XVT's hypertext on-line help system, see the "Hypertext On-line Help" chapter of the *XVT Portability Toolkit Guide*.

4.5.1.1. Fonts and Colors

You can set the control font and the control colors for the entire application, a individual container, or a particular control.



Title: Window 101
Window WIN_101
Help Topic: <Load Help File>
Initial ☐ Invisible ☐ Disabled ☐ Maximized ☐ Iconized
Callback Menubar: None
Type: ● Document ○ Double ○ Plain ○ Modal ☐ Place Exact
Contains ☐ Close Box ☐ Horz. Scrollbar ☐ Vert. Scrollbar
Capabilities ☐ Iconizable ☐ Sizeable ☐ Keyboard Navigation
☒ Own Font ☒ Own Color
X: 133 Width 400
Y: 96 Height 290

Figure 4.9.Window Attributes Dialog (Macintosh platform)

Set Fonts

Tip: To set a font for a particular control,

1. Open the Attributes window.
2. Click on the Own Font box.
3. The Set Font button will be activated.
4. Click on the Set Font button.

The dialog box shown below will appear.

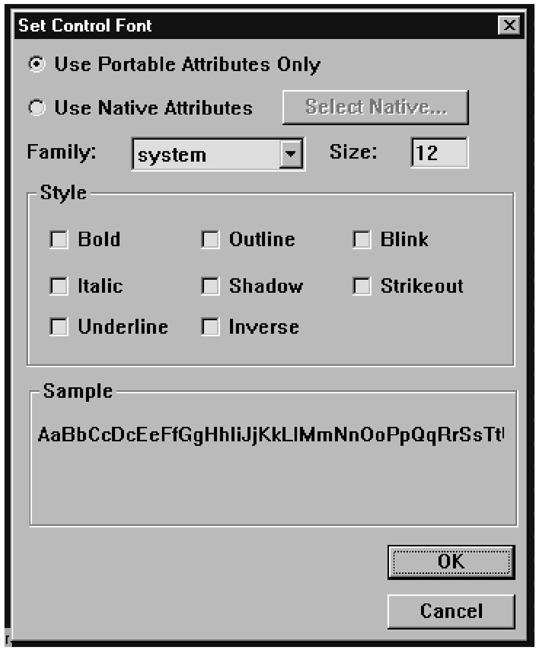


Figure 4.10.Set Fonts Dialog (Win32 platform)

5. To select a portable font, click on any (or all) of the family, size or style options.

To select a platform-specific native font, click on the Use Native Attributes button.

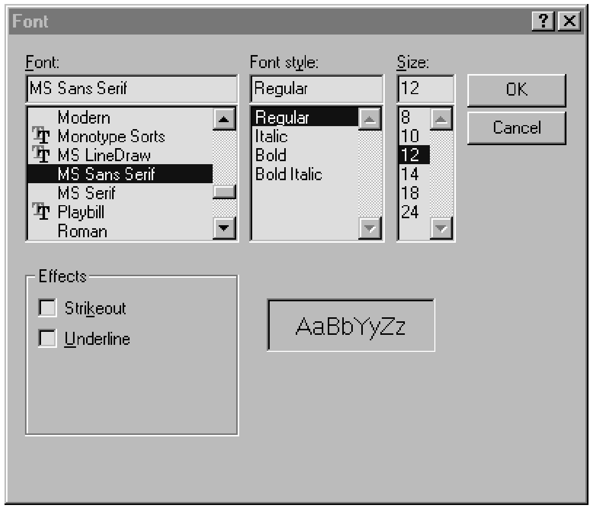


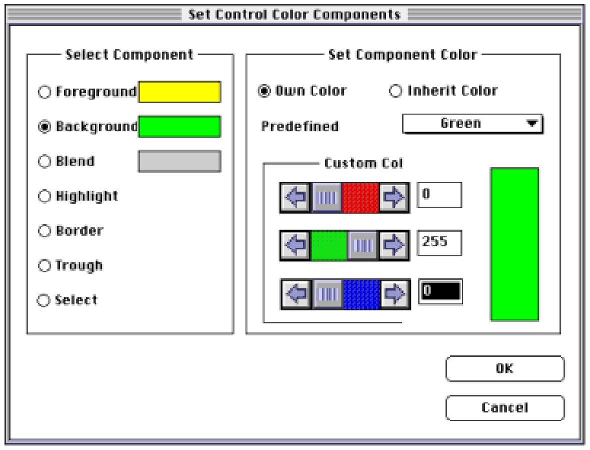
Figure 4.11.Native Fonts Dialog (Win32 platform)

4.5.1.2. Set Colors

Tip: To set a color for a particular control,

1. Open the Attributes window.
2. Click on the Own Color box.

3. Click on the Set Color button.


Set Control Color Components

Select Component

- ○ Foreground
- [● Background]
- ○ Blend
- ○ Highlight
- ○ Border
- ○ Trough
- ○ Select

Set Component Color

[● Own Color] ○ Inherit Color

Predefined: Green

Custom Col: 0 / 255 / 0

Figure 4.12.Set Colors Dialog (Macintosh platform)

Note: In the diagram above, the Highlight, Border, Trough and Select components do not have a color swatch visible to the right of the option. No swatch signifies that the component inherits its color from its parent.

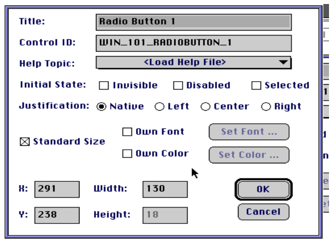
4. Click on the component whose color you want to change.
5. Then click on the Own Color radio button.
6. Then click on the Predefined list box and choose the color you want.

To *create your own* color,

7. Once you have chosen the predefined color, move the scroll bars or enter numbers for the values that comprise the color.

Note: Most systems cannot display all possible colors. XVT recommends the use of predefined colors whenever possible for greatest portability.

4.5.2. Control Attributes



Title: Radio Button 1
Control ID: WIN_101_RADIOBUTTON_1
Help Topic: <Load Help File>
Initial State: ☐ Invisible ☐ Disabled ☐ Selected
Justification: ● Native ○ Left ○ Center ○ Right
☒ Standard Size ☐ Own Font ☐ Own Color
X: 291 Width: 130
Y: 238 Height: 18

Figure 4.13.Attributes Dialog for Controls (Motif Platform)

Standard Size

Sets a control to be the standard height, which is predefined for this type of control on this platform. On each platform, standard size is the native or "natural" size for a control of a certain type.

If the standard size attribute is set, you can change only the width of a control, since the height is predefined. The exception to this is the vertical scrollbar control, where the width is predefined and you can change only the height.

To change the size in both dimensions, turn off standard size. Standard size for a control is set if you simply click to create the new control. If you drag off the Tool Palette to create the control, the standard size attribute is not set. The standard size attribute applies to all controls except list boxes, group boxes, and custom controls.

Default

Sets a push button control to respond to the default choice. The default button is automatically activated when the user presses Return; it is typically titled "OK".

Only one push button can be the default. If you have set the default attribute for one push button, then attempt to set the default attribute for a second push button, the first push button's default attribute will be set to Normal. Setting the default attribute also affects the creation order for controls, as explained later in this chapter. The default attribute applies to push button controls in dialog boxes only.

Cancel

Sets a push button control to respond to the cancel choice. This is the control that is automatically activated (on many platforms) when the user presses Esc; it is typically titled “Cancel”.

If you have set the cancel attribute for one push button, and then attempt to set the cancel attribute for a second push button, the first push button’s cancel attribute will be set to Normal. The cancel attribute applies to push button controls in dialog boxes and modal windows.

Initial State

These checkboxes determine the state of the control when it is created at application runtime.

Invisible
Sets a control to be initially invisible.

Selected
Sets a check box or radio button to be initially selected, or checked. This attribute applies to check box and radio button controls only.

Disabled
Sets a control to be initially disabled. The control is grayed out so that it is unselectable.

See Also: The xvt_vobj_set_* functions, described in the *XVT Portability Toolkit Guide*, are useful for changing the state of controls at application runtime.

Selection

These radio buttons determine whether the application user can make single or multiple selections in a list box, or whether the list box is read-only (i.e. no selections can be made). This attribute applies to list box controls only.

Justification

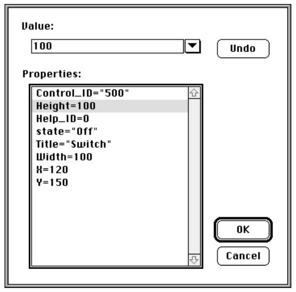
Sets whether a control’s text label will be left-, center-, or right-justified (when possible for a particular toolkit). This attribute defaults to native justification, which tells XVT to use whatever justification is used by default by the native platform. The justification attribute applies to all controls except scrollbars, list boxes, and custom controls.

Note: The conventions and capabilities for control label justification vary from platform to platform. Individual platforms may or may not adhere to this attribute's setting.

4.5.3. Custom Controls

Custom controls do not have a fixed set of attributes. To distinguish them from the standard XVT controls, the attributes of a custom control are referred to as *properties*. Each type of custom control has its own set of properties, although some properties are common to most types of custom controls (such as size and location).

Properties are displayed and changed in the custom controls dialog:



```
100

Control_ID="500"
Height=100
Help_ID=0
state="Off"
Title="Switch"
Width=100
X=120
Y=150
```

Figure 4.14.Custom Controls Dialog

Tip: To set a property's value:

1. Select the name of the property in the list box.
   The current value of the property is displayed in the list edit above the list box.
2. Type the new value for the property in the list edit
   OR
   Choose a new value from the list edit's pull-down list.
   Not all properties of all custom controls have values in the pull-down list. For instance, a size attribute would be unlikely to have a pull-down list of values, since all integers from one to the maximum size of the control would have to be listed.

Tip: To restore the previous value of the property:

1. Click Undo.

Tip: To dismiss the dialog:

1. Click OK to save your changes and dismiss the dialog
   OR
   Click Cancel to dismiss the dialog and discard all of your changes.

At startup time, XVT-Design loads the properties for each custom control from Control Description Files.

4.5.3.1. Control Description Files (CDF)

A CDF file describes the properties that may be set in XVT-Design for a custom control that is created in a window. See *Creating Controls* on page 4-18 for information on how a custom control is created.

File Names

For a custom control named **ccname**, the CDF file must be named ccname.cdf. The **ccname** part of the name becomes the class name. When referring to the custom control name internally, XVT-Design converts it to lower-case. We recommend that you use an entirely lower-case file name on case-sensitive systems like UNIX.

The custom control can also have an image (bitmap) that is used by XVT-Design to render it, at design-time only (not runtime); it must have the name ccname.bmp and be in the same directory as the CDF. Again, the file name should be all lower-case on case-sensitive systems.

File Location

When it starts up, XVT-Design builds a list of all available custom controls by scanning for file names with a .cdf extension. The following directories are scanned:

1. The current directory.
2. The directory containing the configuration (**design.cfg**) file.
3. The directory defined by the XVTCDF attribute, if the configuration file, **design.cfg**, contains such a definition.
4. The directory defined by the XVTCDF environment variable, if the environment contains such a definition.

If duplicate file names are found (using a case-insensitive comparison), the first occurrence is taken.

Custom Controls Without CDF Files

To create a custom control in a window, you are not required to have a CDF file. (This maintains compatibility with earlier versions of XVT-Design.) If you do not have a CDF file, choose "Other..." from the custom control menu. (See *Creating Controls* on page 4-18.)

You will be prompted to enter the class name before you can drag out the custom control in the window. Once a class name is entered in this way, and you place a custom control of that class in a window, it will appear on the custom control menu, although it still won't have a CDF file. You will be able to set values for standard properties (e.g., TITLE and HEIGHT) via the Custom Control Attributes dialog.

To set control-specific attributes for the control, type into the ACE for the tag "SPCL:Instance Data," following the rules for that particular control. If you subsequently provide a CDF file for the control, XVT-Design automatically moves properties defined for the control from the action code for the "SPCL:Instance Data" tag to the Custom Control Attributes dialog when you bring up the Custom Control Attributes dialog. Any properties not defined in the CDF file are left in the action code.

Forming the Attributes for the Custom Control

At runtime, XVT-Design automatically generates appropriate data and code to pass properties and values to the custom control when it is created. From the custom control's perspective, properties and their values are stored in attributes.

The application developer may specify properties in the Custom Control Attributes dialog (those defined in a CDF), as action code for the "SPCL:Instance Data" tag, or in both places. One reason that both places might be used is that some properties may have values (e.g., running text) that do not fit the property/value model used by CDF files.

XVT-Design always places properties and values from the Custom Control Attributes dialog first in attributes, followed by text supplied as action code.

Text entered as action code is not reformatted, and appears exactly as entered, one line of instance data per line of action code (following any CDF properties).

The following standard properties that always appear in the Custom Control Attributes dialog are not also set as instance data, although they are available to the custom control via other means (see XVT Technical Note #148):

Title

Control_ID

X

Y

Width

Height

Format of a CDF File

A CDF file consists of property descriptions, one per property. One property description is separated from the next by one or more blank lines. Comment lines start with a # (in column one) and can appear anywhere.

A property description consists of up to four fields. The first three (Name, Type, and Default) are one per line. The fourth (Description) can be on multiple lines. The fields must be in the order Name, Type, Default, and Description. All but the Name are optional.

Here are details on the fields:

Name

The name of the property, limited to letters (case insensitive), digits, and the underscore.

Type

One of the following: boolean, color, enum, int, or string.

Default

The default value, used when initially populating the Custom Control Attribute dialog.

Description

One or more lines of descriptive text, up to 500 characters in length, that could be displayed when the property is selected in the Custom Control Attributes dialog. (This feature is not currently implemented, so no Description text is displayed and the field may be entirely omitted.)

The type (e.g., string) can be optionally followed by a colon and a comma-separated list of values used to populate a drop-down list; from this list you can select a value. Or, you can enter the value in the edit control. With the exception of the enum type, an entered value need not match a value in the drop-down list, as long as the value is legal for the type (see next section).

Spaces are ignored within a CDF line, except within a Default value or Description text.

Property Types

The type of a property determines the values that you can enter in the edit field within the Custom Control Attributes dialog. Validation of the value occurs when the value is accepted in the dialog, not while the value is typed, (i.e., when another property is selected or the dialog's OK button is clicked). These types and their allowed values are:

boolean

A Boolean value, which must be TRUE, FALSE, T, or F (case insensitive).

color

A color value, either three decimal numbers (0 through 255) separated by commas that give values for the red, green, and blue components, or one of the following symbols (case insensitive): RED, GREEN, BLUE, CYAN, MAGENTA, YELLOW, BLACK, DKGRAY, GRAY, LTGRAY, or WHITE.

enum

An enumerated value, which behaves like a string value, except that the value must match one of those listed after the type in the property description (e.g., "enum: TOP, LEFT, BOTTOM, RIGHT")

int

An integer value, which consists of a string of digits (0 - 9) optionally preceded by a sign (+ or -).

string

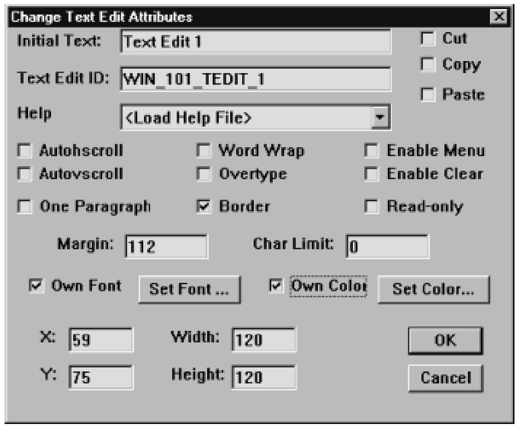
A string value, which consists of up to 255 arbitrary characters.

4.5.3.2. Example CDF

```
#  CDF for Calendar custom control.
#
VIEW
enum: DAILY, WEEKLY, MONTHLY
MONTHLY

BACK_COLOR
color
GRAY

TEXT_COLOR
color
BLACK

RULE_COLOR
color
BLUE

START_YEAR
int
1980

END_YEAR
int
2010

HEADING
string
LANG

enum: English, French, Spanish, German
English
MSG

string: Choose date, Pick date, Select date, Click on choice
Choose date
```

4.5.4. Text Edit Attributes

The following attributes apply to text edit objects only. For more information regarding text edit objects, see the *XVT Portability Toolkit Guide*.


Change Text Edit Attributes

Initial Text: Text Edit 1

Text Edit ID: WIN_101_TEDIT_1

Help: <Load Help File>

☐ Cut
☐ Copy
☐ Paste

☐ Autohscroll
☐ Autovscroll
☐ One Paragraph
☐ Word Wrap
☐ Overtype
☑ Border
☐ Enable Menu
☐ Enable Clear
☐ Read-only

Margin: 112
Char Limit: 0

☑ Own Font
☑ Own Color

X: 59
Y: 75
Width: 120
Height: 120

Figure 4.15.Attributes Dialog for Text Edit Objects (Win32 Platform)

Initial Text

The text shown in the text edit object when it is first displayed at application runtime. (This attribute is the same as the Title attribute for other objects.)

Autohscroll

Enables automatic horizontal scrolling of the text edit object when the user drags the mouse outside of the view rectangle.

Autovscroll

Enables automatic vertical scrolling of the text edit object when the user drags the mouse outside of the view rectangle.

One Paragraph

Limits the entered text to one paragraph. A paragraph is terminated with a carriage return. When this attribute is set, the paragraph is limited to the number of characters entered in the Char Limit field.

Word Wrap

Keeps words together and wraps them to the next line when there is not enough room on the first line (as determined by the number of pixels specified in the Margin field).

Overtype

Determines whether the text edit object is initially in overtype mode. In this mode, typed characters overwrite existing text. If this attribute is not set, the text edit object defaults to insert mode.

Border

Places a border around the text edit object.

Cut

Allows users to cut text from the text edit object.

Copy

Allows users to copy text from the text edit object.

Paste

Allows users to paste text into the text edit object.

Enable Menu

Determines whether the text edit system will enable and disable the commands on the Edit menu of the window that contains the text edit object. Do not check this attribute if the window's menubar does not have an Edit menu.

Enable Clear

Causes the Clear menu item to be always enabled when the text edit object is active.

Read-only

Sets the text in the text edit object to be read-only (unselectable).

Margin

Sets the maximum number of pixels allowed per line when Word Wrap is enabled, thereby determining the right margin.

Char Limit

Sets the limit for the number of characters that can be entered in the text edit object, if the One Paragraph attribute has been set.

4.5.5. Dialog Box Attributes

Type: Modal/Modeless

This control sets the type of the dialog box being created as either modal or modeless.

Invisible

Sets the dialog to be initially invisible. It can be made visible in the application by calling the XVT function xvt_vobj_set_visible.

Disabled

Sets the dialog to be initially disabled. It can be enabled in the application by calling the XVT function xvt_vobj_set_enabled.

Callback Class

Specifies the class name for a dialog. If you enter a name in a dialog's callback class edit field, or select a name from the callback class list, XVT-Design generates function calls for all of the dialog's tags, with names based on the class name. This lets you easily use classes of functions defined in external files.

The function calls are inserted in each tag's action code, unless you have already entered code for that tag. XVT-Design will not overwrite code that you have entered before you set the callback class of a dialog.

Own Color

Allows you to change the colors of the dialog.

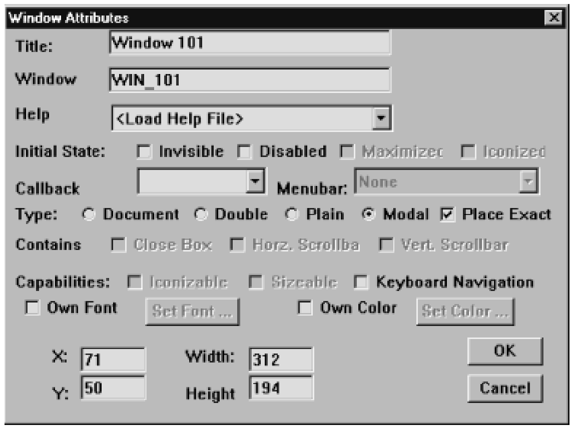
Own Font

Allows you to change the fonts of the text in the dialog.

Example: Suppose you set the callback class attribute of a dialog to be "MyCl". In the action code for the dialog's Create event tag, XVT-Design puts the following function call:

```
xd_dlg_MyCl_Create(xdWindow, xdEvent, 0L);
```

4.5.6. Window Attributes



Title: Window 101
Window WIN_101
Help <Load Help File>
Initial State: ☐ Invisible ☐ Disabled ☐ Maximizec ☐ Iconized
Callback Menubar: None
Type: ○ Document ○ Double ○ Plain ● Modal ☑ Place Exact
Contains ☐ Close Box ☐ Horz. Scrollba ☐ Vert. Scrollbar
Capabilities: ☐ Iconizable ☐ Sizeable ☐ Keyboard Navigation
☐ Own Font ☐ Own Color
X: 71 Width: 312
Y: 50 Height 194

Figure 4.16.Attributes Dialog for Windows (Win32 Platform)

Callback Class

Specifies the callback class name for a window. If you enter a name in a window's callback class edit field, or select a name from the callback class list, XVT-Design generates function calls for all of the window's tags, with names based on the class name. This lets you easily use classes of functions defined in external files.

The function calls are inserted in each tag's action code, unless you have already entered code for that tag. XVT-Design will not overwrite code that you have entered before you set the callback class of a window.

Example: Suppose you set the callback class attribute of a window to be "MyCl". In the action code for the window's Update event tag, XVT-Design puts the following function call:

```
xd_win_MyCl_Update(xdWindow, xdEvent, 0L);
```

Initial State:

Invisible

Sets a window to be initially invisible. It can be made visible in the application by calling the XVT function xvt_vobj_set_visible.

Disabled

Sets a window to be initially disabled. It can be enabled by the application by calling the XVT function xvt_vobj_set_enabled.

Maximized

Sets a window to be initially maximized.

Type

Sets the window's type. There are three types of borders available: Document, Double Border, and Plain border.

Modal

Modal windows are windows that force the user to address an issue raised in the window before continuing.

Place Exact

If the Modal radio button is clicked, the **Place Exact** box becomes available. The Place Exact button sets the WSF_PLACE_EXACT flag in the attribute flag of a window creation call. This function allows you to overwrite default window behavior that a particular platform might enforce.

Colors/Fonts

Control colors and fonts can be specified at the window level.

See Also: For details on how to specify a color or font, see *Fonts and Colors* on page 4-26.

4.5.6.1. Attributes that Affect Only Document-Type Windows

The following attributes affect only windows with their type attribute set to Document.

Close Box

Determines whether a window's decoration includes a close box.

Iconized

Sets a window to be initially iconized (minimized). This affects only applications that run on the Win, PM, and Motif platforms.

Maximized

Sets a window to be initially maximized. Motif platforms ignore this setting.

Note: The Iconized and Maximized attributes are mutually exclusive—setting one clears the other.

Capabilities

These checkboxes affect the border controls on the window. Checking the attribute's box indicates that the control is present.

Iconizable

Determines whether a window can be iconized. This affects only the Win, PM, and Motif platforms.

Sizeable

Determines whether a window's decoration includes size controls.

Vertical Scrollbar

Determines whether a window includes a vertical scrollbar. XVT-Design knows that scrollbars are included in windows; activating this button allows you to attach HScroll attribute to the window.

Horizontal Scrollbar

Determines whether a window includes a horizontal scrollbar. XVT-Design knows that scrollbars are included in windows; activating this button allows you to attach VScroll attribute to the window

4.6. Specifying Creation Order

On the Edit menu, the Creation Order command lets you specify the creation order for controls within the currently active dialog box or window. When a user navigates an application by pressing keys, the creation order for controls determines the order in which the controls are traversed. Also, the creation order determines which of one or more overlapping controls will be drawn "in front" of the others.

Tip: To view or edit the creation order for controls:

1. Select a window or dialog box.

2. From the Edit menu, choose Creation Order.
   A dialog box shows the control names and their current traversal order.

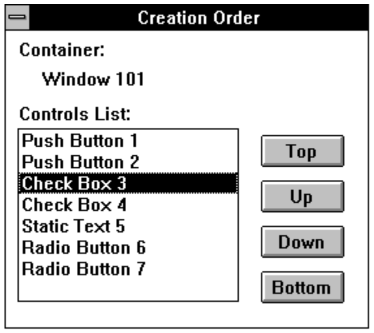


Figure 4.17.The Creation Order Dialog (Windows Platform)

The current traversal order reflects the order in which you created the controls, except for two cases:

- Any push button control with the Default attribute automatically appears in the first position
- Any push button with the Cancel attribute automatically appears in the second position.

Tip: To rearrange the current order:

1. Select the name of the control to be moved.
2. Click the Up, Down, Top, or Bottom button to move the control to a new position.

Up, Down
Moves the selected control name one position up or down.

Top, Bottom
Moves the selected control name to the top or bottom of the list.

Note: If any push buttons have the Default or Cancel attribute set, another control cannot move to the top of the list.

Tip: Selecting a control in the Creation Order dialog also selects the control in the layout window. This is useful for finding controls that have been obscured by other controls. Also, double-clicking on a control's name in the Creation Order dialog opens the control's attributes dialog.

4.6.1. Keyboard Navigation in Windows

Keyboard navigation is the use of keyboard input instead of mouse pointing and clicking to interact with GUI objects. Generally, native look-and-feel for keyboard navigation includes using the **Tab** key and **Shift-Tab** key (back-tab) to traverse through controls in a window or dialog. Alternatively, the user may type character keys (associated with **mnemonic characters**) to select an object directly. A mnemonic character is preceded by a tilde (~) in the title text and displayed with an underline to users. ("Title text" refers to the title field of the control attribute dialog.) Groups of controls (such as radio buttons) may be traversed with **Arrow** keys.

Unlike XVT *dialogs* which automatically provide keyboard navigation to users, XVT *windows* require special handling to implement keyboard navigation. The XVT_NAV navigation object encapsulates the navigation list of controls, child windows, and custom controls for a particular window. The navigation object allows you to specify the navigation order for your application's windows. Any control mnemonic character set in the control's title will be processed automatically on the XVT/Win32 platform where control mnemonics are supported in native look-and-feel.

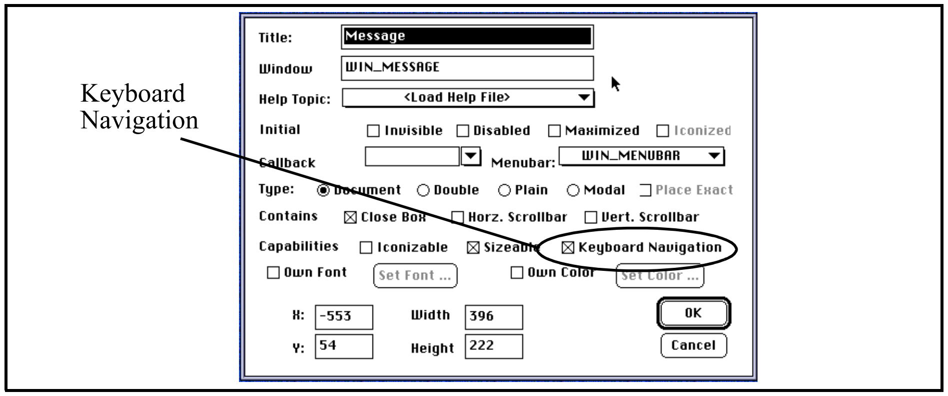
Navigation is provided by checking the "Navigation" checkbox in the window attribute editor. You do not need to create the XVT_NAV navigation object.

See Also: For detailed information about the xvt_nav_* functions, refer to their descriptions in the *XVT Portability Toolkit Guide*.

4.6.1.1. Add Keyboard Navigation to a Window

To add keyboard navigation to a window,

1. Open the window's Attributes dialog.
2. Click on the Keyboard Navigation button.



Title: Message
Window: WIN_MESSAGE
Help Topic: <Load Help File>
Initial: ☐ Invisible ☐ Disabled ☐ Maximized ☐ Iconized
Callback: Menubar: WIN_MENUBAR
Type: ● Document ○ Double ○ Plain ○ Modal ☐ Place Exact
Contains: ☒ Close Box ☐ Horz. Scrollbar ☐ Vert. Scrollbar
Capabilities: ☐ Iconizable ☒ Sizeable ☒ Keyboard Navigation
☐ Own Font Set Font ... ☐ Own Color Set Color ...
X: -553 Width 396
Y: 54 Height 222
OK Cancel

Figure 4.18.Keyboard Navigation box

4.6.2. Radio Button Groups

Radio buttons, by definition, are combined into groups. Only one radio button in a group can be checked at a time, allowing the application user to choose one of several alternatives. XVT-Design automatically generates code to handle radio button groups.

A group of radio buttons consists of two or more radio buttons with consecutive creation order within a container (window or dialog). If one container has two or more groups, the groups must be separated in the creation order list by at least one control of any type other than a radio button.

The position of the radio buttons within a container has no effect on their grouping. The position of other controls in the container, including group boxes, does not affect radio button groups either. Only the creation order of the controls dictates the grouping of radio buttons. This allows you to lay out the controls without restriction.

4.6.2.1. Creating and Using Radio Button Groups

Since radio button groups are determined only by creation order, you can lay out your controls with XVT-Design first, then determine their grouping later. Of course, you should group radio buttons visually as well as functionally, so that your application user understands how to operate the controls.

Group boxes and static text controls are useful for separating radio button groups because either the group box title, or the static text, can describe the radio button group to the user. At the same time, you can use the group box or static text to programmatically separate the group from other groups, by placing it after the radio group in the creation order of the controls. Use the Creation Order command on the Edit menu to adjust the creation order of the controls.

Example: The following figure shows a dialog box, with two radio button groups, as it would appear in an executing application:

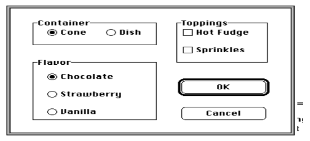


Figure 4.19.Dialog with Radio Button Groups (Macintosh Platform)

The first group contains two radio buttons, while the second contains three. The user can choose one of two kinds of containers, and one of three flavors. Unlike the radio buttons, the Toppings check boxes do not operate as a group—one, both, or neither can be checked.

The next illustration shows the creation order for the controls in this dialog:

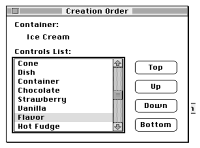


Figure 4.20.Creation Order for Radio Buttons (Macintosh Platform)

Notice that the group boxes (Container and Flavor), which visually surround and delimit the radio button groups, follow the radio buttons in the creation order list. Because of their position in the creation order of the dialog's controls, they functionally separate the two radio button groups.

4.6.2.2. Responding to Radio Button Events

XVT-Design creates code for radio button groups when you generate source files for your application. When the application user clicks a radio button in a group, this code checks that button and unchecks the previously checked button. You do not have to write any code to implement this behavior.

Although XVT-Design generates code that implements the visual behavior appropriate for radio button groups, you must write the code that responds to the E_CONTROL events generated by the application when the user manipulates the radio buttons.

4.7. Using the Menu Editor

With XVT-Design, you can design multiple menubars for your application. Each menubar can have hierarchical menus that descend from it. To help you understand menus in XVT, here are some basic definitions:

Menubar
A menubar is the "root" of the menu hierarchy tree. To design menus, you must start with a menubar. A menubar, which consists of a list of menus, is visually represented by a row of names across the top of a screen or window.

Menu (also called a **pull-down menu**)
Menus appear horizontally across a menubar. When you click on a menu (or select it via a keyboard mnemonic), it "pulls down" a vertical list of items for you to choose from. A menu can contain submenus.

Menu item
Menu items appear on a menu or submenu. A menu item can be a "leaf" of the menu tree, in which case it causes an E_COMMAND event to be delivered to the application. Or, it can be another submenu whose contents are displayed when the user drags the mouse to this item.

Hierarchical menu
A hierarchical menu has one or more submenus. Such a menu/

submenu arrangement is hierarchical because it can contain several nested levels of menus.

Submenu

A submenu is just like a menu, except that it can appear anywhere in your menu hierarchy. When a submenu appears as an item on a menu (or submenu), some graphical indication—such as an arrow—is used to show that the menu hierarchy extends below this item. When the user pulls down a menu and moves the mouse to a submenu, the list of menu items for that submenu appears.

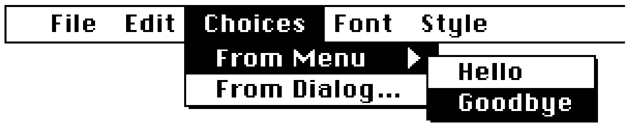


Figure 4.21.Hierarchical Menu with Submenu (Macintosh Platform)

In the illustration above, the menubar consists of the File, Edit, Choices, Font, and Style menus. The Choices menu is a hierarchical menu; its submenu is titled "From Menu". The From Menu submenu has two menu items, titled "Hello" and Goodbye".

4.7.1. Menubar Editor

You can create new menubars and edit their menu hierarchies by using the Menubar Editor.

Tip: To invoke the Menubar Editor:

From the Tools menu, select Menubar Editor.
The Menubar Editor dialog appears.

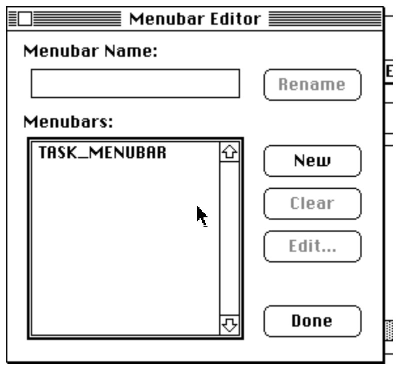


Figure 4.22.The Menubar Editor (Macintosh Platform)

The Menubar Editor consists of a dialog containing a list box, an edit control, and several push buttons. The list box shows the names of all menubars in your project. To select a menubar, click on its name. The push buttons apply different actions to the selected menubar.

The Menubar Editor dialog contains the following controls:

Rename

Changes a menubar's symbolic identifier. To do this, select the menubar in the list box and type the new name in the edit control. Then click the Rename button.

New

Creates a new menubar. The name of the new menubar is added to the list.

Clear

Deletes the selected menubar. The menubar is permanently removed from the project, and it is not copied to the clipboard.

Edit

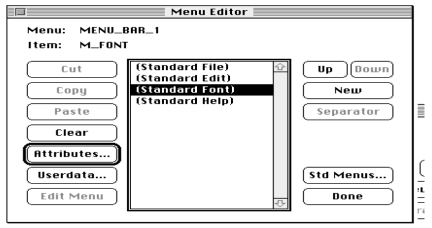
Invokes the Menu Editor for this menubar (see "Menu Editor" below).

Done

Dismisses the Menubar Editor.

4.7.2. Menu Editor

When you invoke the Menu Editor for a new menubar, a list box appears containing the four standard menus: File, Edit, Font, and Help. The menus are listed in left-to-right order.



Menu: MENU_BAR_1
Item: M_FONT

Figure 4.23.The Menubar Editor (Macintosh Platform)

Cut

Deletes the selected menu or menu item and puts it on the clipboard.

Copy

Copies the selected menu or menu item to the clipboard.

Paste

Inserts the contents of the clipboard after the selected menu or menu item.

Clear

Deletes the selected menu item without putting it on the clipboard, thereby permanently deleting it.

Attributes

Invokes the Menu Attributes dialog, where you can set various attributes for a menu, menu item, or submenu (see “Menu Attributes”).

Userdata

Invokes the Userdata Editor, where you can associate up to six strings of arbitrary text with a menu, menu item, or submenu.

Add Menu/Edit Menu

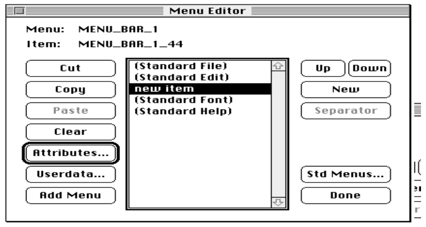
Invokes the Menu Editor again, so you can descend to the next level below the selected menu item. This button reads “Edit Menu” if the object selected in the list box has a submenu, and reads “Add Menu” otherwise.

Up/Down

Moves the selected menu or menu item either one position up, or one position down in the current menu. (If you are editing a menubar, moving a menu up corresponds to moving it left on the menubar.) Note that you cannot position any items in front of the Standard File or Standard Edit menus, and you cannot position any items after the Standard Font or Standard Help menus.

New

Creates a new menu or menu item, inserting it after the currently selected menu item.


Menu Editor

Menu: MENU_BAR_1
Item: MENU_BAR_1_44

(Standard File)
(Standard Edit)
new item
(Standard Font)
(Standard Help)

Figure 4.24.Menu Editor Showing a New Menu (Macintosh Platform)

Separator

Inserts a separator bar after the currently selected menu item. You can insert separators only into menus and submenus; you cannot insert them on the menubar.

Std (Standard) Menus

Brings up a dialog from where you can select which standard menus you want. The standard menus (File, Edit, Font, Help) can be selected only for a menubar, not its submenus.

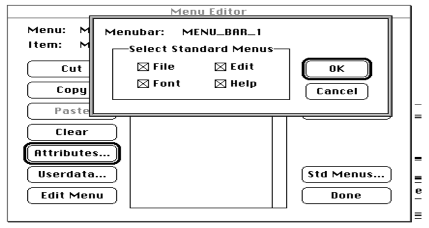


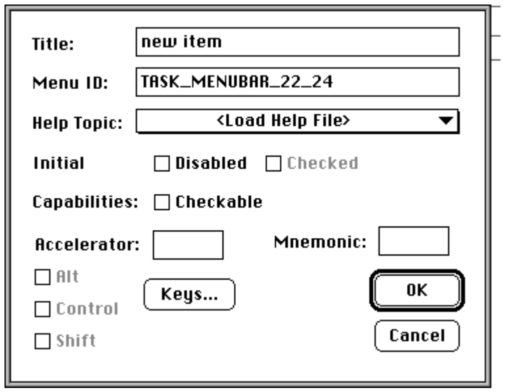
Figure 4.25.Dialog for Selecting Standard menus (Macintosh Platform)

Done

Dismisses the Menu Editor dialog. You can also dismiss the dialog by clicking its close box.

4.7.3. Menu Attributes

When you click Attributes for a menu or menu item, the Menu Attributes dialog is displayed.


Title: new item

Menu ID: TASK_MENUBAR_22_24

Help Topic: <Load Help File>

Initial: ☐ Disabled ☐ Checked

Capabilities: ☐ Checkable

Accelerator:

Mnemonic:

☐ Alt

☐ Control

☐ Shift

Figure 4.26.The Menu Attributes Dialog (Macintosh Platform)

The controls in the Menu Attributes dialog are described below. When you invoke the dialog for a menu (rather than a menu item), all of the controls except Title, Menu ID, Mnemonic, and Help Topic are disabled.

Title

The text for this menu or menu item.

Menu ID

The symbolic identifier for this menu or menu item.

Mnemonic

Specifies the one-character mnemonic you wish to associate with this menu or menu item. At run-time, the mnemonic can be used on those platforms that support keyboard navigation of menus. The mnemonic character must be one of the characters in the title of the menu or menu item.

Accelerator

Specifies the one-character accelerator key that you wish to associate with this menu item. At run-time, the accelerator can be used as a substitute for selecting the menu item with the mouse.

Keys

Brings up the Accelerator Keys dialog, where you can specify a function key as the accelerator for this menu item. When you choose one of the keys, its full name is inserted in the Accelerator edit field.

Alt/Control/Shift

If you have selected an accelerator for this menu item, you can specify the modifier keys (Alt, Control, or Shift) to be used with the accelerator key.

Checkable

Specifies that this menu item can be checked.

Checked

Specifies that this menu item should initially be checked.

Disabled

Specifies that this menu or menu item should initially be disabled. The item can be enabled by the application with the XVT API function xvt_menu_set_item_enabled.

4.8. String Resources

4.8.1. Strings

In XVT-Design, you can create and manipulate strings and string lists, to be used as resources from within an XVT program. The advantage of string resources is that you can maintain strings outside your executable program. As a result, you can modify them without having to recompile the program.

Strings and string lists are useful for adding text to your application that would not otherwise be in its resources. For instance, you could use a string list to initialize a list box or list button. String resources are also useful for holding text that may change when the application runs. As an example, you could have a button with the title “Find”, that changes to “Find Again” at some point during execution. The second string, “Find Again”, could be stored as a string resource.

Tip: To create a string:

1. From the Tools menu, choose Strings Editor.
   The Strings dialog box appears.

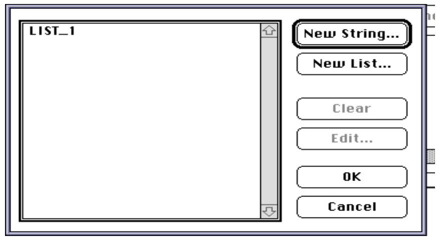


Figure 4.27.The Strings Dialog (Windows Platform)

2. Click New String.
   A new string is created, whose contents are initially “New String”. The String Edit dialog box appears, in which you can change both the string and the symbolic identifier.
3. Change the string and/or its symbolic identifier, and click OK.

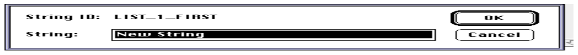

String ID: LIST_1_FIRST
String: New String

Figure 4.28.The String Edit Dialog (Windows Platform)

In your application, add a call to xvt_res_get_str to retrieve the string at run time. The function prototype for xvt_res_get_str is:

```
char *xvt_res_get_str(int rid, char *s, int sz_s)
```

For the first argument, simply use the symbolic identifier for the string you want (in the above example, you would use STR_ALIGNRT). You must allocate a buffer large enough to hold the string and pass the address of the buffer, as well as the buffer’s size, as the second and third arguments.

4.8.2. String Lists

String lists are useful if you have a list of names or labels that you would like to enumerate outside of your application. Let’s say you had a list containing the names of all the states in the United States. In your application, you could then retrieve the list as a whole, and populate a list box with the resulting list.

Tip: To create a string list:

1. From the Tools menu, choose Strings Editor.
2. In the Strings dialog, click the New List button.
   A new string list is created, and the String List dialog appears.

Example: In the String List dialog box, the symbolic identifier, which you can change, appears at the top. For example, you might change this to be "SL_USA").

When you click the New String button, a new entry with the contents "New String" is created, and the String List Edit dialog appears. Change the contents of the string (perhaps to something like "Arizona"), and click OK.

Click New String again (which creates another "New String"), and change it to "Alabama". Clicking New one more time creates another string that you could change to "Georgia".

To arrange your strings in ascending alphabetical order, you can reorder your string list. To reorder an item, select the one you would like to reorder (in our example, "Alabama"), and click Up. Alabama is now in the first position, and Arizona in the second. You could have selected any item and used the Up and Down buttons to change its position in the string list.

The result is a string list whose symbolic identifier is "SL_USA", and whose contents are the three strings "Alabama", "Arizona", and "Georgia". To retrieve the entire string list into your application program, call xvt_res_get_str_list, whose function prototype is the following:

```
SLIST xvt_res_get_str_list(int rid_first,
        int rid_last)
```

The two arguments to xvt_res_get_str_list are the symbolic identifiers for the list ("SL_USA"); "_FIRST" and "_LAST" are appended respectively. Your call to xvt_res_get_str_list would therefore look something like this:

```
SLIST x;
x = xvt_res_get_str_list(SL_USA_FIRST, SL_USA_LAST);
```

Following the call to xvt_res_get_str_list, the contents of the SLIST x will be the strings you created using XVT-Design.

4.9. Userdata Strings

XVT-Design's Userdata feature lets you associate arbitrary data (up to six text strings) with any control, dialog box, window, or menu you create. For instance, you could associate textual data with a dialog box control.

Example: Imagine an application in which a user could perform various queries on a separate database program by pressing buttons in a dialog box. With XVT-Design's Userdata feature, you could associate the query commands needed for the database with a particular control in the dialog box. Then you could write C code that would send the associated userdata strings (the query commands) to the database, whenever the control was selected.

Userdata is stored in the application's URL file. As a result, you can change the userdata for an application without recompiling the application.

XVT-Design lets you create and edit userdata to be associated with an object. You can also change the labels that are associated with the userdata items themselves.

The XVT API provides three functions for accessing the userdata you have created with XVT-Design:

- xvt_res_get_dlg_data
- xvt_res_get_win_data
- xvt_res_get_menu_data

See Also: See the *XVT Portability Toolkit Guide* for more details.

4.9.1. Creating Userdata

There are two ways to invoke the Userdata window for creating and editing userdata strings:

- Select an object (window, dialog, or control) in a layout window, and choose Userdata from the Edit menu
- Select a menu or menu item in the Menu Editor, and click the Userdata button

In the Userdata window, you can create or modify six different userdata strings. These are the userdata strings that will be associated with the object you have selected.

Note: Before using the Userdata command on a window or dialog box, make sure that no controls are currently selected within the window or dialog box.

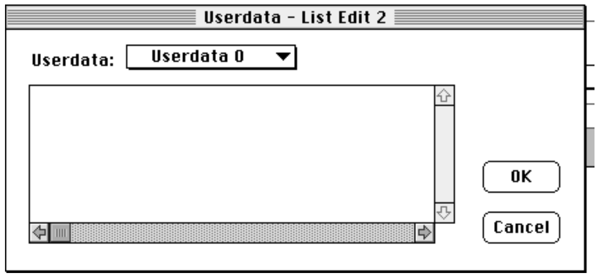


Figure 4.29.The Edit Userdata Window (Macintosh Platform)

4.9.2. Editing Userdata

The Userdata window contains a list button and a scrollable text edit pane. The list button contains the labels of the six userdata strings (see below). To choose one of the six userdata strings for editing, select the appropriate label from the list button.

The text edit pane contains the text of the userdata string indicated by the list button. You can enter an unlimited number of lines of userdata for each userdata string. Userdata can include newlines, which show up as “\n\” in the URL file.

4.9.3. Userdata Labels

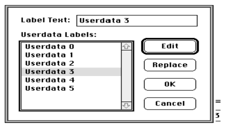
In the Edit Userdata window, userdata labels are associated with userdata strings. Userdata labels are simply titles or descriptions of each of the six userdata strings that can be associated with objects. The default userdata labels are “Userdata 0”, “Userdata 1”, ... “Userdata 5”.

Userdata labels only appear in XVT-Design’s Edit Userdata window. They are not placed in the generated source code or URL files. The labels are just to remind you what a particular userdata string means.

You can create labels before you have created a window, dialog box, menu, or control. You can also change the userdata labels even if you have not created any of these objects. You can edit userdata labels without having to edit the userdata itself.

Tip: To edit userdata labels:

1. From the Edit menu, choose Userdata Labels.
   The Edit Userdata Labels dialog box appears.


Label Text: Userdata 3

Userdata Labels:

Userdata 0
Userdata 1
Userdata 2
Userdata 3
Userdata 4
Userdata 5

Figure 4.30.Edit Userdata Labels Dialog (Macintosh Platform)

2. In the Edit Userdata Labels dialog, select a label in the list box.
3. Click Edit (or double-click the label).
   The label appears in the edit field.
4. Edit the label, then click Replace to change the label.

Edit

Puts the selected label into the edit field.

Replace

Replaces the existing label with your new label.

OK

Accepts the changes that you have made to the labels and removes the dialog box. After you click OK, any future Edit Userdata window that you bring up displays the new userdata labels above each of the six userdata strings.

Cancel

Removes the dialog box without applying the changes you have made to the userdata labels.

4.9.4. Generating Code with Userdata

When you generate an application, the userdata is written to the project URL file as a userdata statement in a dialog, window, or menu definition. If the userdata is associated with a control, the userdata URL statement follows the control definition in the URL file.

4.10. TestMode

XVT-Design's TestMode lets you verify the appearance of your application's user interface without compiling or linking. TestMode emulates your application's appearance at runtime, as if you had compiled, linked, and executed it.

During TestMode, XVT-Design's user interface is replaced with your application's. Instead of seeing XVT-Design's windows and dialogs, you'll see those of your project.

In TestMode, your project's menus, windows, dialogs, and controls behave as if your application were actually running. If you induce any events for which you have defined connections (for example, choosing menu items or manipulating controls), the actions for those connections are executed. You can open windows by choosing menu items, create or destroy dialogs by clicking push button controls, and so forth.

If you use XVT-Design on more than one platform, you can use TestMode to check your application's appearance on each platform. XVT-Design's project files are portable across all supported platforms.

Note: Keep in mind that XVT-Design only keeps track of connections in TestMode. It does *not* interpret or execute your program's code. TestMode ignores any code you have entered with the ACE. You do not need to generate any source files before using TestMode. The presence or absence of generated files has no effect.

4.10.1. Entering TestMode

Tip: To test your project:

1. From the Tools menu, choose Enter TestMode.XVT-Design hides its menubar and any open windows and dialogs. The task window and its menubar are replaced with your project's task window and menubar. A special "TestMode" menu is appended to the right of your application's menubar.
2. Test any menus, windows, dialogs, and controls in your project for which you have defined connections.

When you enter TestMode, your application receives a Create event. If you want a window or dialog to appear as soon as your application starts, create a connection for the application's Create event tag to create the window or dialog.

The special TestMode menu appears on all of your application's menubars while you use TestMode, to provide a way for you to exit TestMode. It does not appear in your compiled, stand-alone application.

Note: If you haven't created any connections in your project, TestMode is not very interesting. You will see only your application's task window (on platforms that have a task window) and its menubar. Choosing menu items won't have any effect, except choosing Quit from the File menu, which will cause XVT-Design to leave TestMode.

4.10.2. Leaving TestMode

Tip: To leave TestMode:

1. From the TestMode menu in your application's task window, choose End TestMode.
   -OR-
   From your application's File menu, choose Quit or Exit.
   -OR-
   Close your application's task window, if the native window system provides a way to do this.

Note: If you have defined a connection for the Quit item on a File menu, choosing this item will *not* terminate TestMode.

When you leave TestMode, XVT-Design redraws its menubar and reopens any windows that were open before you entered TestMode.

4.10.3. Special Considerations for TestMode

Although XVT-Design emulates your application's runtime appearance and behavior as accurately as possible, there are a few limitations.

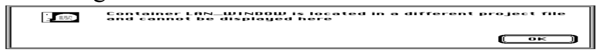
4.10.3.1. About Box

In TestMode, you will not see your application's About dialog box. A dummy dialog box is displayed in its place.

4.10.3.2. External Connections

If you have any connections to objects outside of the current project, XVT-Design will not be able to invoke these objects during

TestMode operation. If your application attempts to execute such a connection during TestMode, you will see a dialog like the following one:



Container LAN_WINDOW is located in a different project file and cannot be displayed here

Figure 4.31.External Connections in TestMode (Macintosh Platform)

To resolve connections to objects external to your current project, you must merge your project file with the project file that contains the external objects.

See Also: Refer to the "Project File Management" chapter of this manual for more information on external objects and merging project files.

4.11. Generating Source Code

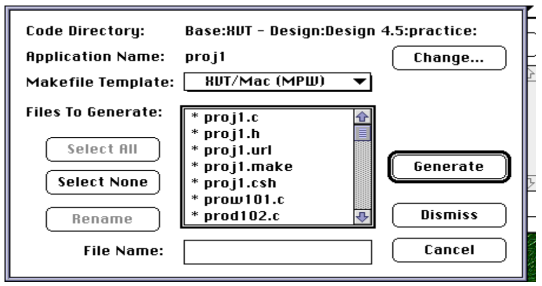
Once you've created the windows, dialog boxes, menus, and other resources for your project with XVT-Design, the next step is to generate the source code for your application. XVT-Design automatically generates Universal Resource Language (URL) files for your resources, and C source code files to handle these resources in your final application.

4.11.1. Setting the Destination Directory

By default, XVT-Design puts the files it generates in the directory where the project file was opened from or the current directory.

Tip: To change the destination directory:

1. From the File menu, choose Generate Application.
   The Generate Application dialog appears.


Code Directory: Base:XVT - Design:Design 4.5:practice:

Application Name: proj1

Makefile Template: XVT/Mac (MPW)

Files To Generate:
```
* proj1.c
* proj1.h
* proj1.url
* proj1.make
* proj1.csh
* prow101.c
* prod102.c
```
File Name:

Figure 4.32.The Generate Application Dialog (Macintosh Platform)

The current destination directory is displayed at the top of the dialog.

2. Click the Change button. A standard save-file dialog appears.
3. Navigate to the directory in which you want XVT-Design to place your application's source code files.
4. Click the OK button.

Note: This procedure does not actually create any files; it simply sets the destination directory for when the files are generated.

4.11.1.1. The Application Name

The name of your application is distinct from the name of its project file. For instance, the name of your project file might be "**MDB30.DPR**", but the application it generates could be named "**MegaDB_3.0**".

You set the name of your application when you set the destination directory, as described in the previous section. Before clicking the Save button in the save-file dialog, type the name of your application in the filename edit control in that dialog.

4.11.2. Filenames

XVT-Design creates names for the files it generates, based on several pieces of information:

- the name of the application (which you supply)
- the type of the file

- the resource associated with the file

In most cases, the first three characters of each file's name will be the same as those of the application's name.

In the filenames given in the following sections, we indicate these first three characters as "**<xxx>**", and the complete application name as "**<appname>**". For example, if your application is named "hello", the filename described as "**<xxx>appl.c**" would actually be "**helappl.c**".

You can change these names as you desire. See "Choosing Files," later in this chapter.

4.11.3. Types of Generated Files

XVT-Design generates several different kinds of files for your application: C source code files, a header file, a resource file, a help text file, and a makefile. The following sections describe these files and their naming conventions.

4.11.3.1. C Source Code Files

XVT-Design creates a separate **.c** file for each module in your project (including the application itself). These files contain event handlers for the module, and any code fragments that you created with the Action Code Editor.

The names for these files have the following format:

<xxx><r><nnn>.c

where "**<xxx>**" is the first three characters of the application's name. "**<r>**" is a character indicating the type of the associated resource, which is interpreted as follows:

d - dialog box

m - menu

w - window

"**<nnn>**" is the ID number of the resource (three digits). Resources are numbered consecutively, in order of their creation.

The **main** function and the Task window's event handler are placed in a file called **<appname>.c**, where **<appname>** is the name of your application.

Example: Suppose your application's name is "hello", and it has one window and one dialog. The source code file for the window will be named **helw101.c**, the file for the dialog will be named **held102.c**, and the main file for the application will be named **hello.c**.

4.11.3.2. Header File

XVT-Design creates one header file, named **<appname>.h**.
It contains function prototypes and resource ID definitions.

4.11.3.3. Resource File

XVT-Design generates one XVT Universal Resource Language (URL) file, named **<appname>.url**. It contains resource descriptions for all of the user-interface objects in your application.

4.11.3.4. Help Text File

XVT-Design puts all text you enter in the SPCL:Help tag in a file named **<appname>.csh**.

Tip: This feature is provided primarily for compatibility with XVT-Design Release 2. For convenience, XVT recommends that you do not use this feature to enter and edit help source text. Instead, use an external text editor or authoring tool.

4.11.3.5. Makefile

XVT-Design creates a complete makefile for compiling and linking your application's source files. It is named **makefile.<xxx>**.

4.11.4. Choosing Files to Generate

The first time you generate source files for a given project, you will probably want to generate all of these files. Later, when you modify the project (for instance, to change the items in a menu, or add a new dialog box) you may not need to regenerate all the files.

Tip: To choose which files to generate:

1. From the File menu, choose Generate Application.
   The list box in the dialog lists all the files that are part of your application, and whether or not they are to be generated. If an asterisk ("*") precedes the filename, XVT-Design will generate a new copy of that file. If no asterisk precedes the filename, XVT-Design will not generate a new copy of that file.
2. To turn a file's asterisk off or on, double-click the filename.

3. To enable generation of all of the files, choose Select All. To disable generation of all of the files, choose Select None.

Tip: To change the name of a generated file:

1. In the list box, click on the file's name.
   The name appears in the edit control at the bottom.
2. In the edit control, change the file's name.
3. Click the Rename button.

4.11.5. Makefiles

XVT-Design creates makefiles by using pre-defined template files for each compiler and operating system configuration supported by XVT, adding information about the particular files for your project when it generates the makefile.

If there are no pre-defined templates that suit your system configuration, you can modify the templates using any text editor.

Note: To change the template ID, you can modify an existing template file. To create a new template file, modify an existing one.

Tip: To choose a makefile template for your project:

1. From the File menu, choose Generate Application.
   The Makefile Template list button lists all of the available templates.
2. Choose the appropriate template from the list button.

You can generate a makefile using any available template, regardless of the platform on which you run XVT-Design.

If none of the templates supplied with XVT-Design suit your development environment, create your own templates by editing copies of the most-suitable supplied templates. Make sure you modify the name of the template in the TPL file so that you can identify your template in the XVT_Design list of makefiles.

4.11.5.1. How XVT-Design Finds Makefile Templates

XVT-Design locates makefile templates by searching for files with the filename extension **.tpl**. It searches for makefile templates in the following directories, in the order they are listed here:

- The current directory
- The directory that contains the **design.cfg** configuration file

- The value of XVTTPL, if XVTTPL is a macro defined in **design.cfg**, and is defined as a valid directory.
- The value of XVTTPL, if XVTTPL is a variable defined in your environment, and is defined as a valid directory.

4.11.5.2. External Files

Source-code files that are part of your application's code, but not generated by XVT-Design, are called external files. External files are usually used for portions of your program other than the user-interface code.

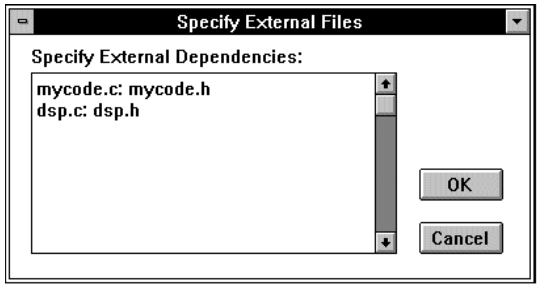
You can add external file references to your XVT-Design project. When XVT-Design generates the makefile for your application, it inserts dependency information for your external files.

Tip: To add external file references:

1. From the File menu, choose External Files. The External Files dialog appears.
2. Type the names of your external files in the dialog. Place each filename on a separate line. Each file can depend on one or more other files. Dependencies are described as follows:

   <dependent_filename>: <filename1> <filename2> ...

Example: In the following illustration, **mycode.c** and **dsp.c** are external files: **mycode.c** depends on **mycode.h**, and **dsp.c** depends on **dsp.h**.



```
mycode.c: mycode.h
dsp.c: dsp.h
```

Figure 4.33.Dependent External Files (Windows Platform)

4.12. Code Recovery

With XVT-Design you can edit the generated files using a text editor and later recover code from inside of XVT-Design.

If you enabled Code Recover, then code fragments were wrapped with special comments during the code generation process. You can edit the files using a standard text editor; then click on a "Recover Code" button to recover all the changes you have made to the generated files.

Note: Code Recovery is an optional feature. You must click on the Code Recovery button in the Project Attributes dialog (which activates the feature for the entire application) *before the files are generated* in order to recover code afterwards.

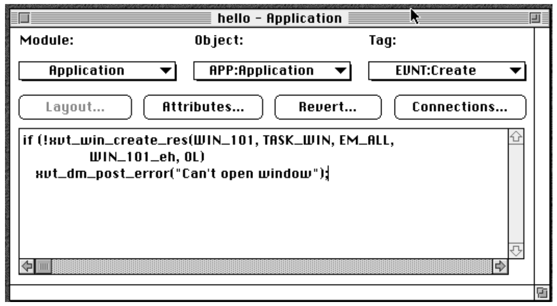
The code recovery dialog is similar to the code generation dialog; it allows you to select which files you want to recover.

4.12.1. Edit and Recover Code

The following example illustrates how a simple code fragment can be recovered.

Example:

1. Assume the following ACE code fragment:



```
if (!xvt_win_create_res(WIN_101, TASK_WIN, EM_ALL,
          WIN_101_eh, 0L)
  xvt_dm_post_error("Can't open window");
```

Figure 4.34.ACE window with sample code

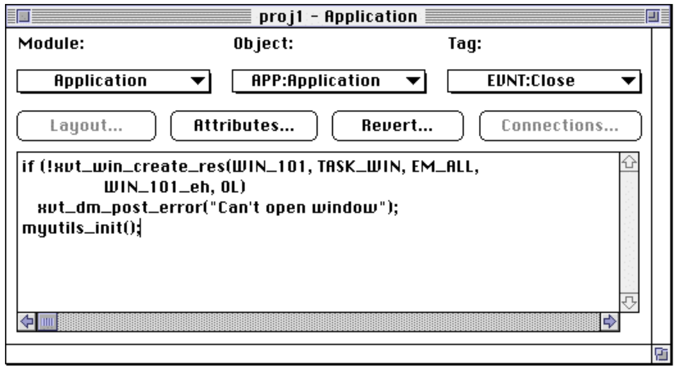
2. The ACE code is framed with special comments during the code generation process:

```
...
task_eh(WINDOW xdWindow, EVENT *xdEvent)
...
    case E_CREATE:
    ...
        /*RECOVERABLE CODE BEGIN Create */
        if (!xvt_win_create_res(WIN_101,
            TASK_WIN,EM_ALL, WIN_101_eh, 0L)
            xvt_dm_post_error("Can't open window");
        /*RECOVERABLE CODE END Create */
    break;
...
```

3. Edit the generated files using a text editor. (In this example, we have added an additional piece of code calledmyutils_init(). You can modify as well as add code.)

```
...
task_eh(WINDOW xdWindow, EVENT *xdEvent)
...
        case E_CREATE:
        ...
                /*RECOVERABLE CODE BEGIN Create */
                if (!xvt_win_create_res(WIN_101,
                        TASK_WIN,EM_ALL, WIN_101_eh, 0L)
                        xvt_dm_post_error("Can't open window");
                        myutils_init();
                /*RECOVERABLE CODE END Create */
        break;
...
```

4. When you select the Recover Code option from the File menu in the ACE, XVT-Design will recover all of the changes you have made to all the generated files:



```
if (!xvt_win_create_res(WIN_101, TASK_WIN, EM_ALL,
          WIN_101_eh, 0L)
  xvt_dm_post_error("Can't open window");
myutils_init();
```

Figure 4.35.ACE window with modified code

4.12.2. Important Notes About Recovering Code

- Only code between the Begin and End comments is recovered.
- Do not modify the Begin or End comments in the generated files.
- Remember: this is an *optional* feature. You must click on the Recover Code checkbox in the Project Attributes dialog box *before* you generate the application in order to recover code later.

- Unlike ACE code, you can edit the generated files with a text editor and you see exactly where your code is in the generated files, which helps fix regular errors detected when you compile.

4.12.3. Special Caution When Using the Code Recovery Feature

If you have previously generated code with Code Recovery enabled, you **must** recover code **before** you modify your project with XVT-Design. The reason for this requirement is that the code recovery algorithm depends on the current state of the project. For example, if a control is added to a window, then an additional tag will be expected during code recovery.

Caution: **XVT-Design will not recover any code if the project and generated files get out of sync**.

If you inadvertently modify your project prior to recovering code, you **must** get your project in sync with the generated files.

- The best way to get your project in sync is to undo whatever changes you made to the project.
- A second method is to duplicate any edits you made to the generated files in the ACE.
- A third way is to manually modify the tags in the generated files to reflect the current state of your project.

In general, follow these steps:

1. Modify your project using XVT-Design
2. Generate code
3. Edit tags in generated files
4. Recover code
5. Repeat the process

5

INTERNATIONALIZING YOUR APPLICATION

5.1. Introduction

5.1.1. About Internationalization and Localization

This section highlights some of the general issues involved in adapting applications for international language and locale support.

5.1.1.1. Why and When to Adapt an Application

Adapting an application for a specific locale, or *localization*, involves several issues. You must evaluate the differences in locales to determine which, if any, locale categories are relevant to the application. For example, written language (and character codesets) may be a minor issue to some localization efforts. An application developed for American users will have only slight differences for British citizens (date formatting, monetary formatting and minor variations in spelling). In other cases, all locale categories may be affected by a localization effort. An Asian language-based application will have significantly different needs than the same English-based application including such categories as character codesets, layout, collation, and monetary formatting. Localization, and to what degree localization is performed, are strongly dependent on the target locales of the application.

Several important tasks are involved in the localization of an application. Often, the most daunting task is the translation of string literals from English (or another original language; this guide assumes that English is the base language) to the local language. Often this task is outside the scope of the application developer and will require you to obtain specialized expertise. Secondary to this task is proper processing of those strings whether that be sorting, concatenating, parsing, or simply formatting strings for screen layout. Other related tasks include creating locale-specific resource files and setting the environment configuration appropriate to the locale.

Internationalization (I18N) involves modifying application code and resource files so they can be easily localized. Ideally, the result of I18N is that localization efforts can be accomplished without changing application source code and without requiring code recompilation. Applications that target a significant variety of locales are candidates for internationalization. In deciding whether to internationalize your application or not, you must evaluate the pros and cons that arise from supporting several versions of an application, each for a specific locale, or even a single version which contains code for multiple locales. Also, it is important to consider which locales you may need to support in the future.

There are several things you need to consider before you internationalize an application. In addition to handling single-byte character strings, your application code must be able to process multibyte or wide character strings depending upon the locales you must support. This includes general processing for collation, parsing, formatting and layout. String literals and other locale-sensitive items should be made external to the application source code so they can be translated and substituted as needed.

5.2. Internationalization Support In XVT PTK

The XVT PTK has been engineered for internationalization. All of the XVT PTK functions support both single-byte and multibyte character sets, and XVT utilities such as **curl** accept multibyte input. All internal PTK strings and resources are externalized for easy localization, and XVT now provides the localized resources for French, German, Italian and Japanese.

See Also: For detailed information on internationalization features of the PTK, see Chapter 19 of the *XVT Portability Toolkit Guide.*

5.3. Internationalization Support in XVT-Design

XVT-Design can generate internationalized applications. When the project **Internationalize** attribute is selected, Design will generate all locale-specific information in internationalized form, and all string processing code emitted by Design will be multibyte ready.

Attribute strings passed to Custom Controls will also be internationalized.

The internationalization scheme used to extract locale-specific strings for XVT-Design generated code can also be used to internationalize user code. Specific information on using this capability follows.

XVT-Design itself has not been internationalized. XVT-Design does not accept multibyte input, and cannot be used to directly generate a multibyte application. It can, however, generate an internationalized application that can be efficiently localized to a multibyte language outside of XVT-Design.

5.4. The LOCAL_C_STRING Macro

Note: Note: for a step-by-step summary of how to use this feature, see *A Step-by-Step Guide to Internationalization* on page 5-8 below.

XVT-Design defines a family of macros designed to allow flexibility and user configurability in generating internationalized code. It generates all local-specific information in the form of calls to these macros. For example, where in non-international mode XVT-Design would generate the call:

```
xvt_dm_post_note("Can't open file");
```

in international mode it will generate the call:

```
xvt_dm_post_note(LOCAL_C_STR(xd_LS_cant_open_file,
        "Can't open file", xdStrBuf1, xdBUFSZ));
```

Note that the arguments to LOCAL_C_STR include both the original literal string (the one XVT-Design would generate in non-international mode) and a symbolic name that can be used to identify the locale-specific translation of the string that will appear in the final internationalized application (In this case the identifier name resembles the English string that XVT-Design knows of, but this need not always be true.)

XVT-Design will generate the following definition for LOCAL_C_STR:

```
#define LOCAL_C_STR(id, literal, buf, bufsz) \
        xvt_res_get_str(id, buf, bufsz)
```

Note that the literal string passed to LOCAL_C_STR is ignored, and the symbolic identifier is used as a string resource ID. The reason for including the literal string in the LOCAL_C_STR macro arguments is discussed in the next section.

There are several advantages to this style of code generation:

- You can use the LOCAL_C_STR macro to internationalize your own code.
- XVT-Design can use the LOCAL_C_STR macro in the default ACE code written for connections, for example.
- You can redefine the LOCAL_C_STR macro and implement a different internationalization scheme than the one XVT-Design uses by default. LOCAL_C_STR can also be used to temporarily disable internationalization during application development. (See discussion of XVT_LOCALIZABLE below.)

Note: **Note:** In most cases XVT-Design uses the ID of a menu, window or control as a base to form the symbolic identifier argument to the LOCAL_* macros. When using international mode, you should make sure your menu, window and control ID's are no more than 26 characters long.

5.5. Using LOCAL_C_STR in User Code

The LOCAL_C_STR macro can be used anywhere in C code where function calls are allowed. The LOCAL_C_STR macro will evaluate to a character pointer.

When writing a LOCAL_C_STR macro invocation, you will need to choose a symbolic identifier for the macro's first argument. XVT recommends using a name that corresponds closely to the literal string in the second argument. This will aid in the localization task later.

The second argument, a string literal, is used when internationalization is turned off (See discussion of XVT_LOCALIZABLE below) and by strscan to create a file of locale-specific information to be translated (See strscan below). It should be in the same single-byte language you do your primary XVT-Design development work in.

The third and fourth arguments are a pointer to a buffer that will hold the string as it is fetched from resources and the size of the buffer in bytes. XVT-Design generates definitions for four buffers, xdStrBuf1 through xdStrBuf4, all of size xdBUFSZ, that can be used for this purpose, but remember that XVT-Design will also generate code that uses the same buffers. In general, if you want the string to persist past other calls to LOCAL_C_STR, you should allocate your own buffer and pass it to LOCAL_C_STR.

Note: **Note:** You may use the same symbolic identifier more than once as long as the corresponding literals are identical. Since strscan ignores `#if` and `#ifdef` preprocessor commands, do not use the same symbolic identifier with different literals in opposite branches of these conditionals.

5.6. Other LOCAL_* Macros

XVT-Design defines and uses several other LOCAL_* macros, all of which address different internationalization problems. They are not generally as useful in user code as LOCAL_C_STR.

The LOCAL_URL_STR and LOCAL_URL_RECT macros allow XVT-Design to extract locale-specific information from the generated application URL file. XVT-Design will generate calls to these macros for you.

LOCAL_URL_STR and LOCAL_URL_RECT are different from LOCAL_C_STR in that there is no need to extract the locale-specific information to resources - these calls *occur* in resources. It is still helpful, however to textually separate the locale-specific information in the URL so it may be maintained separately

The LOCAL_C_STR_INIT macro is used by XVT-Design to internationalize the property lists passed to Custom Controls. It is also generated automatically where appropriate and its use in user code is not recommended.

5.6.1. LOCAL_* Macro Definitions

To see how XVT-Design actually defines the LOCAL_* macros, select **Internationalize** in your application's Project Attributes dialog and generate your application. The LOCAL_* macros will be defined in your Design-generated application header file.

5.7. The XVT_LOCALIZABLE Compile-Time Flag

You may want to do initial application development concentrating on basic functionality, and internationalize only when your application is nearly complete. Or you may want to turn internationalization off while debugging or adding functionality to a mature internationalized application. To allow for these needs XVT-Design defines the XVT_LOCALIZABLE compile-time flag.

With XVT_LOCALIZABLE turned off, the LOCAL_* macros are defined to return their literal arguments, and internationalization is effectively disabled. You should select the **Internationalize** project attribute as early as possible in your development cycle, and keep XVT_LOCALIZABLE turned off until you are ready to deal with internationalization issues. In the meantime, XVT-Design will know to generate LOCAL_C_STR macros in your default connection code.

5.8. The strscan Utility

Note: Note: for a step-by-step summary of how to use this feature, see *A Step-by-Step Guide to Internationalization* on page 5-8 below.

The LOCAL_* macros effectively remove the locale-specific information from your code. The next task is to provide the locale-specific information elsewhere so the application is complete.

Ideally, all the locale-specific information for you application would be isolated in one text file. It would be then be possible to give just this one file to a translator. It should also be possible to maintain this translation separately from the XVT-Design project file and the application external C files without having to redo all the translation work when minor changes are made to the application user interface.

Fortunately, the design of the LOCAL_* macros makes this possible, and XVT provides a utility, **strscan**, that automates much of the process.

strscan scans C and URL files for LOCAL_* macros, and writes an URL file that contains all the locale-specific information from the macros and can easily be translated. For example, when strscan encounters the example code from above:

```
xvt_dm_post_note(LOCAL_C_STR(xd_LS_cant_open_file,
    "Can't open file", xdStrBuf1, xdBUFSZ));
```

It writes the following URL code:

```
STRING xd_LS_cant_open_file "Can't open file"
```

When this URL statement is translated, compiled and loaded in your application, the LOCAL_C_STR macro definition will cause the translated string to be loaded from resources.

Note: **Note:** Actually, **strscan** writes two output files. One, named strres.h by default, contains the locale-specific information as described above and is designed to be included in your application's Design-generated URL file. The other, named strdef.h by default, contains defines that give the STRING resources of strres.h numerical resource ID values; it should be included in your application's header. strdef.h contains no locale-specific information and need not be translated.

5.8.1. Using strscan

To create definitions for all your locale-specific information, run strscan on all your C files (both XVT-Design generated and external) and your XVT-Design generated application URL file.

5.8.1.1. strscan Options

`-f filename`	option file (defaults to strscan.opt)
`-r filename`	output resource file (defaults to strres.h)
`-d filename`	output definition file (defaults to strdef.h)
`-n start`	starting number for RID's (defaults to 29999)
`-i increment`	increment for RID's (defaults to -1)
`-e newname macroname`	define synonym for macro
`filename ...`	files to process

These options are valid for both command line and GUI interface versions of strscan.

5.9. Localization

Localization is accomplished by maintaining a translated version of **strres.h** for each locale you wish to support.

After translating your locale-specific strings and recompiling your resources, you will probably find that many of your controls need to be resized to accommodate translated strings of different length than the originals. XVT-Design generates most URL rectangles through the LOCAL_URL_RECT macro for this reason, and you can see the rectangle definitions in strres.h.

It will be necessary to identify the rectangles you need to modify in your translated version of strres.h and manually adjust them. You will need to recompile your resources and run your application to check your work.

5.10. A Step-by-Step Guide to Internationalization

5.10.1. General Steps To Internationalize Your XVT Application

1. In the Project Attributes dialog, select **Internationalization**. This selection causes SPCL:I18N_Header, SPCL:I18N_URL, and SPCL:I18N_Main tags to be created (as described below), as well as inserting special localization macros.

2. In the SPCL:I18N_URL tag in the ACE, you now see code similar to the following:

   ```
   #ifdef XVT_LOCALIZABLE
   #include “strres.h”
   #endif
   ```

   The URL include file **strres.h** contains locale-specific strings; it is generated in step 1 (on page 5-9 below) when you run the **strscan** utility program.

3. In the SPCL:I18N_Header tag in the ACE, you now see code similar to the following:

   ```
   #ifdef XVT_LOCALIZABLE
   #include “strdef.h”
   #endif
   ```

   The include file **strdef.h** will be generated in step 1 (on page 5-9 below). **strdef.h** contains #defines for resource IDs used in **strres.h**.

4. Still in the ACE, replace string literals in your code with calls to the LOCAL_C_STR macro. Use the XVT-Design **Find** command to help you locate string literals.

5. Using the SPCL:User_Header tag in the ACE, add the following code:

   ```
   #define XVT_LOCALIZABLE
   ```

   Alternatively, you may modify your makefile or makefile template to define this flag.

6. Use the XVT-Design **Generate Application** command to generate all files.

Now, working outside of XVT-Design, complete these two additional steps:

7. In your external files (those not generated by XVT-Design), replace string literals with calls to the LOCAL_C_STR macro.

Once you have completed the steps presented in the preceding list (steps 1 through 7), your C application is modified to use the LOCAL_C_STR macro and your application is “internationalized.” In other words, your application’s displayable strings have been processed in a manner that allows them to be easily “localized,” that is to say, modified for a specific locale.

5.10.2. General Steps To Localize Your XVT Application

1. Execute the **strscan** utility on all of your ***.c** and ***.url** files to generate the include files **strres.h** and **strdef.h**. If you have carefully followed steps 4 through 7 (on page 5-8), **strres.h** now contains all your locale-specific strings. View both files after running the utility.
2. Make copies of **strres.h** and give them names that co-workers will recognize as locale-specific resource files, such as **engres.h** and **gerres.h**. You will want to adopt a file naming convention for your different versions of **strres.h**. Renaming the files protects you in the future when you run **strscan**, since **strres.h** is consistently and predictably overwritten when it already exists.
3. Using the SPCL:I18N_URL tag in the ACE, replace the reference to **strres.h** with references to a file of strings translated into German (for example), **gerres.h**, and another file of English strings, **engres.h**. When the editing in your application resource file is complete, this section of code will resemble the following:

```
#ifdef XVT_LOCALIZABLE
#ifdef LANG_GER_W52
        #include "gerres.h"
#else /* English */
        #include "engres.h"
#endif
#endif
```

If you are supporting multiple languages in other localized files, modify the above code as needed to reference these files, as well.

For a list of standard locale names (e.g., LANG_GER_W52), see Chapter 19 of the *XVT Portability Toolkit Guide*.

4. Translate the strings in the locale-specific resource files, such as **gerres.h**, for the locales you need to support.
5. Consider redefining the way dates or money variables are displayed (to match local practices). Likewise, in your external files (those not generated by XVT-Design), search for all sprintfs that you wish to format for locale-specific display. For more details and an example, refer to Chapter 19 of the *XVT Portability Toolkit Guide*.
6. Compile your resources and check the translation of text and the size and position of GUI objects.
7. Adjust the size and positions defined by creation rectangles in **strres.h** to accommodate the increased or decreased lengths of the translated strings.

You do not need to re-translate your entire **strres.h** file when you make changes to your application. Usually it is only necessary to regenerate **strres.h** and **strdef.h** using **strscan**, then identify the strings that have been added or changed and add their translated equivalents to your translated versions of **strres.h**.

Building the locale-specific executable requires the setting of one or more specific #defines. XVT source code files are "localized" when XVT_LOCALIZABLE is defined, and switch to a specific language based upon other #defines, as well. To build the locale-specific executable, follow these additional steps:

8. Modify your makefile or makefile templates to build localized versions of your resources. If you wish to build, for example, a German version, you would also define LANG_GER_W52.

   Refer to the example at the end of this section for an example of how to modify a UNIX makefile. Different programmers or organizations have their own personal preferences and different platforms will require slightly different syntax.

On some platforms, you may need to run **curl** manually from the command line, as shown in the following **curl** compile statement:

```
curl -r rcwin -i..\..\include -dLANG_GER_W52
        -dLIBDIR=.\..\..\lib app.url
```

Although the command line shown above is printed on two lines, you should enter a command line as a single line.

You now have a resource file—if you view it, you will see, in this case, that all strings are now in German.

9. If your makefile did not completely finish the build, you should now complete any unfinished steps in your build process.

Example: This example shows a UNIX makefile that builds a German version of an XVT application:

```
# Define localized options.
# Start a German build.

LOCALIZE_OPTS        = -dLANG_GER_W52
CC_OPTS              = -c $(INC_PATH)
CURL                 = $(XVT_DSC_DIR)/bin/curl
...

#
# Include the defines in all source code compilations
.c.o
        $(CC) $(CC_OPTS) $(LOCALIZE_OPTS) $<
# Also pass them to curl
app.uil: app.url
                $(CURL) $(CURL_OPTS) $ (LOCALIZE_OPTS) app.url
...
```

5.11. Advanced Internationalized Topics

You may wish to instrument your application to load the appropriate locale-specific resources at startup time based on an environment variable or options file. This will likely involve some platform-specific code, and you may want do the locale determination in a platform-specific manner.

You will also want to build with the appropriate localized PTK resources and help text. XVT provides these for US English, French, German, Italian and Japanese, and you can translate them to other locales yourself.

For information on these topics, and for an in-depth discussion of issues you will want to consider when internationalizing and localizing your user code, see Chapter 19 of the *XVT Portability Toolkit Guide.*

6

PROJECT FILE MANAGEMENT

If you are working with other developers on the same XVT-based application, you might find it useful to break your XVT-Design project file into several parts.

Working with separate project files allows individual members of the development team to work on their project components in their own files, independently of the other members. Later, you can merge the separate parts back into one file.

The XVT Development Solution for C package provides a utility program for splitting and joining project files called **pfm**. On all platforms (except the Macintosh), this program will run both as a GUI and from the command line.

6.1. Using pfm at the Command-Line

pfm is the project-file splitting and merging utility. You might use **pfm** in scripts you write to help manage concurrent development efforts.

6.1.1. Splitting Project Files

pfm can create several project files, each with only one container, from one project file.

Tip: To split one or more project files into several project files:

Use a command of the following form:

```
pfm -s <source-pf> <basename>
```

where <source-pf> is the original project file to split and <abasename> is used to create names for the new project files. One file is created for the strings, string lists, and menubars, and one file is created for each container in the file <source-pf>.

When **pfm** splits one project file into several, the first new project file contains all of the strings, string lists, and menubars present in the original file, but no containers. Each succeeding new project file contains a single container and only the menubar associated with the container.

6.1.2. Merging Project Files

Tip: To merge several project files into one project file:

Use a command of the following form:

```
pfm -m <output><file0> ... <fileN> >
```

<output> is the name of the new project file that contains all of the containers in files <file1> through <fileN>. <file1> through <fileN> are the project files to be merged.

When **pfm** merges several project files into one, it copies the strings and menubars from the first project file on the command line to the output file. Any strings and menubars in the succeeding project files are *not* copied to the output file.

Caution: Since only the strings and menubars in the first project file are copied to the output file, any changes you make to the strings or menubars in the other project files will *not* be reflected in the output file. Hence you should edit the strings and menubars only in the first project file.

6.2. Using the GUI Version of pfm

The interactive version of pfm allows you to create new project files and copy and paste containers between project files. Also, you can split one file into several single-container files with one command.

Each open project file appears in a separate window. A scrolling list box in the window lists all of windows and dialogs in the project file, by resource identifier and name.

Creating Project Files with pfm

Tip: To create a new project file with no resources:

Choose New from the File menu.

Copying and Moving Containers

Tip: To copy or move one container from one project file to another:

1. Open the source and destination project files.
2. Choose the container by clicking its name in the project window.
3. Choose Copy from the Edit menu to copy the container, or choose Cut to move the container.
4. Click the destination project file's window to bring it to the front.
5. Choose Paste from the Edit menu.

Copying and pasting a container between project files also copies the menubar associated with the container (if there is one).

You can also copy and paste several containers at once. Shift-click to select more than one container in the container list, before copying and pasting.

Splitting and Merging Project Files

Tip: To split a project file into several single-container project files:

1. Open the project file.
2. Choose Split from the Project menu. A dialog box prompts you for the base name of the new files.
3. Type the base name of the new files and click OK. The project files are created with sequentially numbered names. The files are named <obase>1.dpr, <obase>2.dpr, and so on, where <obase> is the base name you supplied in the previous step.

Tip: To merge several project files into one file:

1. When launching **pfm**, add the names of the project files to the command line, for example:
   ```
   pfm proj000.dpr proj002.dpr newdlg.dpr
   ```
2. **pfm** opens and presents a dialog asking you to confirm that the files should be merged. Click the Merge button. (If you click the Open button, the files are opened normally, each in a separate window.) A new project is created, which contains the windows and dialogs from the project files named on the command line. The menubars and strings from the file listed first on the command line are also added to the new project.
3. Choose Save As from the File menu to save the new project file.

Note: This merging method is not available on the Macintosh. Use copying and pasting to create one project file from several other files.

6.2.1. Listing the Project File Containers

Tip: To list the containers located in the project files, use a command of the following form:

```
pfm -l <pf0>...<pfN>
```

6.3. Working with Multiple Projects

Splitting a project file into several smaller project files allows the individual members of a development team to work independently on the same application. Here is a general outline for creating an application with multiple project files:

Create the prototype
Using XVT-Design, one or more developers create a prototype of the application, which contains all of the resources for the application's user interface—the windows, dialogs, menubars, etc. The prototype application is stored in one project file.

Split the project file
Using **pfm**, the prototype's project file is split into several smaller project files.The first such project file contains all of the "shared" resources in the application; i.e., the menubars and strings. The remaining small project files each contain one or more of the windows or dialogs of the prototype.

Distribute and work on the smaller project files
The smaller project files are distributed among the members of the development team. Using XVT-Design, each member refines the layout of the container in their project file(s), and writes code to implement its behavior and interaction with the rest of the application. (As described previously, the strings and menubars should not be modified in the smaller files, but only in the first project file.)

Merge the project files
Again using **pfm**, the separate project files are merged back into one project file, along with the shared-resource project file.

Open the prototype
The merged project file is opened with XVT-Design. XVT-Design generates the source code and resource files for the complete application.

Using Source-Control Systems

You may find it useful to use a source-control system to facilitate shared access to the project files during development. However, such a system must be able to manage binary (i.e., non-text) files, since XVT-Design's project files are not purely ASCII text. (Alternatively, you could use a text-encoding utility such as **uuencode** to convert XVT-Design's project files into text-only files that can be accepted by a source-control system.)

6.3.1. External Connections

If you split your application's project file into several files, you can create connections between objects in different files by using external connections. External connections can be made to objects that are still under development (in other project files), or not yet even created.

External connections are replaced with equivalent regular connections when you merge the project files together with **pfm**.

6.3.2. Name and Identifier Conflicts

If you merge two or more project files that have containers with the same resource identifiers and/or names, the resulting project file will contain conflicting identifiers and/or names. If you generate code from XVT-Design using this project file, your compiler will produce duplicate-symbol errors. Neither **pfm** nor XVT-Design attempts to detect or correct duplicate name conflicts.

Tip: To avoid duplicate names and resource identifiers, adopt some sort of team-wide protocol and conventions for naming new resources.

6.3.3. Merging Unrelated Projects

The pfm may also be used to merge project files which have not been previously split.

Tip: When merging unrelated projects, follow these steps:

1. Create unique menubar names. Rename the conflicting menubars (such as TASK_WIN) in the project files that will be merged.
2. Create unique window IDs. Rename the conflicting windows (i.e. Win_101) in project files that will be merged.
3. Merge projects with pfm.

4. Use XVT-Design to modify the merged project file keeping in mind the following questions:
   - Is the About box set properly?
   - Are all the required external files listed?
   - Are all the required Application:User_Header lines listed?
   - Are all the required Application:User_URL lines listed?
5. Create connections to merged dialogs/windows.
6. Generate code and compile.

7 REFERENCE

7.1. Menu Commands

This section contains brief descriptions of all the commands on XVT-Design's menus. For more detailed descriptions of the dialogs and operations described in this section, see the "Usage" chapter of this manual.

7.1.1. File Menu

The File menu contains commands for manipulating project files, and for generating your application's source code and resource files.

7.1.1.1. New Project

Creates a new project. This project becomes XVT-Design's active project. If other projects are open when you choose this command, XVT-Design hides their windows. See *Project Files* on page 4-1.

7.1.1.2. Open Project

Presents a standard open-file dialog, so you can open a previously saved project file. When you open a project, it becomes XVT-Design's active project. If other projects are open when you choose this command, XVT-Design hides their windows. (A list of open projects is found at the bottom of the Edit menu.) See *Project Files* on page 4-1.

7.1.1.3. Close Project

Closes the active project. If you have made any changes to the project since it was last saved, XVT-Design asks whether you want to save the project before it is closed. See *Project Files* on page 4-1.

7.1.1.4. Save Project

Saves the active project file. The first time you use this command for a new project, a standard save-file dialog appears. You can choose the directory in which to save the file, and enter the project's name. See *Project Files* on page 4-1.

7.1.1.5. Save Project As

Opens a standard save-file dialog, allowing you to choose a new filename and directory for the active project.

Note: If you save your project in a different directory, you should also change the Code Directory setting in the Generate Application dialog.

See *Project Files* on page 4-1.

7.1.1.6. Specify External Files

Opens the External Files dialog, which allows you to add references to source code files *not* generated by XVT-Design to your project file. See *External Files* on page 4-68.

7.1.1.7. Generate Application

Opens the Generate Application dialog, where you can change several things:

- The name of your application, and the directory where XVT-Design places the source code files it generates
- The template file used to create a makefile for building your application
- The names of the source code files and resource files for your application

See *Generating Source Code* on page 4-63.

7.1.1.8. Recover Code

Allows you to recover code even after an application has been generated. There are several steps you must take before performing this action.

See *Code Recovery* on page 4-68.

7.1.1.9. Quit (or Exit)

Closes all open projects and terminates execution of XVT-Design. If any open projects have been changed since they were last saved, XVT-Design asks if you want to save them before they are closed.

7.1.2. Edit Menu

The Edit menu contains commands for editing text and objects, and invoking XVT-Design's various editor windows.

Tip: The toolbar in the layout windows has graphical buttons for rapid access to the Cut, Copy, Paste, and Clear commands on the Edit menu.

7.1.2.1. Cut

Removes the selected text or object and places it on the system clipboard.

7.1.2.2. Copy

Places a copy of the selected text or object on the system clipboard.

7.1.2.3. Paste

Copies the contents of the system clipboard to the window, text edit pane, or edit control that has focus.

7.1.2.4. Clear

Permanently removes the selected text or object.

7.1.2.5. Find

Search through code for specifc text.

7.1.2.6. Find Next

Repeat last text search.

7.1.2.7. Scan Tags

Allows you to quickly review the action code for a portion or the entire project.

7.1.2.8. Select All

Selects all text or objects in the window, text edit pane, or edit control that has focus.

7.1.2.9. Userdata Labels

Opens the dialog for editing userdata label strings. See *Userdata Labels* on page 4-59.

7.1.2.10. Project Attributes

Opens the dialog for setting the project attributes: the About dialog, the task window's menubar, the task and document window titles, and the API name-generation convention. See *Project Files* on page 4-1.

7.1.2.11. Attributes

Opens the attributes dialog for the selected object, or the layout window that has focus. See *Setting Object Attributes* on page 4-23.

7.1.2.12. Code

Opens an Action Code Editor window for the selected object, or the layout window that has focus. See *Using the Action Code Editor (ACE)* on page 4-4.

7.1.2.13. Userdata

Opens a userdata editor window for the selected object, or the layout window that has focus. See *Userdata Strings* on page 4-58.

7.1.2.14. Menu

Invokes the Menu Editor for the menubar associated with a window. If no menu is associated with a window, or if no window is currently active, this menu item is disabled. See *Using the Menu Editor* on page 4-48.

7.1.2.15. Creation Order

Opens the Creation Order dialog, which lets you set the order in which controls in a dialog receive focus during keyboard navigation. See section 4.6 on page 4-43.

7.1.3. Tools Menu

This menu contains commands for opening XVT-Design's various editor dialogs, and for starting TestMode.

7.1.3.1. Action Code Editor

Opens an Action Code Editor dialog. If a layout window has focus, the Action Code Editor's context list buttons will be set to the container, or the selected control. Otherwise, the context list buttons will be set to the application See section 4.2 on page 4-4.

7.1.3.2. Menubar Editor

Invokes the Menubar Editor, for creating and modifying menubars and menus. If the Menubar Editor's dialog is already open, choosing this command brings it to the front. See section 4.7.1 on page 4-49.

7.1.3.3. Strings Editor

Invokes the String and Stringlist Editor, for creating and modifying string resources. See section 7.1.3.3 on page 7-5.

7.1.3.4. External Tool

You can configure this menu item to launch the Image Editor application, for creating and modifying portable image files. See *Appendix A: The Image Editor*.

7.1.3.5. Begin TestMode

Starts XVT-Design's TestMode, so you can examine and demonstrate your application's resources as they will appear at runtime. To leave Test Mode, choose End TestMode from the TestMode menu. See section 4.10 on page 4-61.

7.1.4. Controls Menu

The Controls menu includes commands for creating the various types of controls available in windows and dialog boxes: push buttons, check boxes, radio buttons, scrollbars, static text, edit controls, text edit objects, list boxes, list buttons, list edits, and group boxes. Commands for creating custom controls are located on a submenu of the Controls menu. This menu is available only in layout windows.

Note: Text Edit objects and custom controls cannot be placed in dialogs, so the corresponding commands on the Controls menu are disabled when a dialog's layout window is active.

Tip: To create one or more controls:

1. From the Controls menu, choose the control type.
2. Position the cursor in the upper left corner of the desired location.
3. Either click or drag the control into the desired size.
   If you click to create the control, it will be of the standard size for this type of control.
4. Click and/or drag to create additional controls of this type.
5. To exit from this mode, choose Pointer (or another control) from the Controls menu.

When the Pointer tool is selected, you can select, resize, or move existing controls.

Tip: The palette in the layout windows has graphical buttons for rapid access to all of the controls available on the Controls menu.

7.1.4.1. Custom

This submenu contains the names of all of the custom controls currently available in XVT-Design.

7.1.5. Layout Menu

The Layout menu contains commands for adjusting the position of controls in window and dialog layout windows. This menu is available only in layout windows. If no controls are selected, some of the commands are disabled.

The order in which you select controls determines how the alignment commands will affect the controls. The first control selected is the reference point used to align the other controls.

For example, if you select controls A, B, and C (in that order), and then choose Align Left from the Layout menu, controls B and C will be lined up along the left boundary of control A, because A was the first one selected.

Tip: The toolbar in the layout windows has graphical buttons for rapid access to most commands on the Layout menu.

7.1.5.1. Align Left

Aligns the selected controls along the left boundary of the first selected control.

7.1.5.2. Align Center

Aligns the selected controls along the horizontal center of the first selected control without changing their vertical positions.

7.1.5.3. Align Right

Aligns the selected controls along the right boundary of the first selected control.

7.1.5.4. Align Top

Aligns the selected controls along the top boundary of the first selected control.

7.1.5.5. Align Middle

Aligns the selected controls along the vertical center of the first selected control without changing their horizontal positions.

7.1.5.6. Align Bottom

Aligns the selected controls along the bottom boundary of the first selected control.

7.1.5.7. Even Horizontal Spacing

Adjusts the horizontal position of the selected controls so that they are separated horizontally by the same distance.

7.1.5.8. Even Vertical Spacing

Adjusts the vertical position of the selected controls so that they are separated vertically by the same distance.

7.1.5.9. Make Same Size

Makes all selected controls the same width as the control first selected.

7.1.5.10. Grid

Opens the Grid dialog, where you can set the grid spacing interval, and choose whether the grid is displayed.

7.1.5.11. Hide Toolbar

Hides the toolbar in the front-most layout window, if the toolbar is visible. If the toolbar is already hidden, this command changes to Show Toolbar.

7.1.5.12. Hide Object Palette

Hides the object palette in the front-most layout window, if the palette is visible. If the palette is already hidden, this command changes to Show Object Palette.

7.1.6. Window Menu

The Window menu includes commands for creating new windows and dialogs. When you create a new container, XVT-Design opens a layout window for it. The Window menu also lists all the currently open windows. This menu is updated as you open and close layout windows. Choosing an item from the menu brings the corresponding window to the front. On the menu, any open Action Code Editor windows are listed before the layout windows.

7.1.6.1. New Window

Creates a new window resource in the active project.

7.1.6.2. New Dialog

Creates a new dialog resource in the active project.

7.1.7. Help Menu

The Help menu displays the online Help for XVT-Design. It also give you to access to the online PTK Reference file.

7.2. The Configuration File

XVT-Design uses a configuration file to determine default settings for many user-changeable options. You can edit the configuration file to change these default values, so they suit the characteristics of your development environment and your preferences.

XVT-Design only examines the contents of the configuration file when it starts up. Any changes you make to the file while XVT-Design is running have no effect.

7.2.1. Name and Location

The configuration file is named **design.cfg**. The configuration file can be placed in the same directory as the XVT-Design application or in another directory, depending on the operating system. This section describes these directories, by operating system.

7.2.1.1. Macintosh

On the Macintosh, when XVT-Design is launched, it looks for its configuration file in the folder that contains the XVT-Design application itself.

7.2.1.2. UNIX

Under UNIX, when XVT-Design is launched, it first looks for its configuration file in the current directory. If it does not find the configuration file there, it looks in the user's **HOME** directory.

7.2.1.3. Windows

Under Windows, when XVT-Design is launched, it first looks for its configuration file in the current directory. If it does not find the configuration file there, it looks in the root directory of the volume that contains XVT-Design.

7.2.2. Format

The configuration file is a plain text file. You can use any text editor to create, examine, or change it.

7.2.2.1. Configuration File Options

Commands in the configuration file are placed on separate lines, one entry per line. Entries must be in the following format:

```
<attribute>: <value>
```

where <attribute> is one of the attribute names listed in the "Available Options" section (below), and <value> is an integer, string, or Boolean constant. Booleans are denoted as follows:

```
FALSE:false, f, no, off, 0
```

TRUE:(anything not defined as FALSE)

Uppercase and lowercase are not distinguished in Boolean values.

7.2.2.2. Comments

You can add comments to the configuration file by placing a pound sign, #, at the beginning of the line. All subsequent information on the line is ignored. For example:

```
# This is a comment.
```

7.2.3. Available Options

This section lists all the attributes that can be specified in the configuration file, organized by category.

7.2.3.1. Default Grid Settings

These attributes set the default values for the alignment grids in layout windows. When you create a new window or dialog, its layout window initially has these grid settings. You can change these settings in individual layout windows by using the Grid command on the Layout menu.

gridDisplay

Desc:Shows the layout grid in each layout window

Type:BOOLEAN

Default:FALSE

gridNative

Desc:Use a pixel-based grid (TRUE) or a char-based grid (FALSE)

Type:BOOLEAN

Default:FALSE

gridSnap

Desc:Enables snap-to-grid in layout windows

Type:BOOLEAN

Default:FALSE

gridX, gridY

Desc:Pixel width and height of layout grid

Type:integer

Default:8 (for both)

7.2.3.2. Makefile Template Macros

These options let you create string-substitution macros for your makefile templates.

userMacros

Desc:Allows the user to supply additional information to the makefile generator

Type:string

Default:None
Use this option to declare the names of your macros. You must declare each macro before defining it.

7.2.3.3. File Defaults

backupFileExtension

Desc:Set this macro to "FALSE" to disable the creation of a initial Layout window in a new project.

Type:string

Default:dpb

macCreatorCode

Desc:Configures the file type for generated code

Type:enum (KAHL if ThinkC, MPS if MPW)

Default:MPS

macFileType

Desc:Configures the creator code for generated code

Type:string

Default:TEXT

Default:TRUE

projectFileExtension

Desc:Default project file extension

Type:string

Default:dpr

7.2.3.4. ACE Text Defaults

These options set text characteristics for the Action Code Editor's text pane.

aceFontFamily

Desc:Used to get something other than "System" font in the ACE

Type:enum (FF_FIXED, FF_HELVETICA, FF_TIMES)

Default:None
This option sets the font. The value must be one of the following:

FF_FIXED: a fixed-width (monospaced) font, such as Courier

FF_SANS_SERIF: a plain, sans-serif font, such as Helvetica

FF_SERIF: a serif font, such as Times

The actual font used depends on your platform. If no font is specified, the system font is used.

aceFontSize

Desc:Used to change the point size of the font in the ACE

Type:integer

Default:None

appXDefaultFont

Desc:For XM only. When set to a valid X fontname or alias, assigns the font for controls in XVT-Design layout windows and controls in the generated application. This font becomes the default font for the application and is independent of the font the application uses to draw text in windows. See Exceptions below.

Type:string

Default:None

Exceptions

The default font is not used in the "ghost window" in the generated application or in the emergency output message box displayed with xvt_dm_post_message. If you want to set the default font for the ghost window and emergency output message, then add the file xxinit.c to your application and set the font in the fallback resources in that file. xxinit.c is found in ptk/contrib.

7.2.3.5. Miscellaneous Options

charHeight

charWidth

Desc:Specifies what size character Design should use when it does scaling of windows/controls/dialogs when a Design project file is moved between platforms. If these are not specified, Design chooses a platform-specific character size by which to scale.

Type:integer

Default:None

cfgLogfile

Desc:File into which everything is logged that XVT-Design looks for in the configuration (*.cfg) file

Type:string

Default:None
This option lets you explore how XVT-Design uses the configuration file. If this attribute is set to a filename that can be opened for writing with fopen, XVT-Design creates a text file that lists all inquiries made concerning user-configurable options:

- If an inquiry is made for an option that is specified in the configuration file, an entry of the form "<attribute>:<value>" is added to the log file.
- If the option is not specified in the configuration file, an entry of the form "#<attribute>:" is added. Both of these entries are in the same format as entries for the configuration file itself. You can add options to your configuration file by copying them from the log file.

Note: The log file may contain redundant entries, since XVT-Design makes an entry for each query to a given option.

compileURL (Boolean)
If this attribute is TRUE, XVT-Design will automatically run the **curl** resource compiler to compile your project's resource file when you generate your application's source files.

initialAboutBox (Boolean)

If this attribute is FALSE, XVT-Design does not display its version and copyright dialog box when starting up.

enablePlaceWindowExact

Desc:Enables/disables generation of code that sets ATTR_X_PLACE_WINDOW_EXACT

Type:BOOLEAN

Default:FALSE

enableProjectBackup

Desc:Enables/disables automatic backup of project files.

Type:BOOLEAN

Default:TRUE

externalTool

Desc:Path to external tool invoked by "External Tool..." menu item

Type:string

Default:None

externalToolMenuItem

Desc:Customizes the text of the "External Tool..." menu item

Type:string

Default:External Tool...

initialAboutBox

Desc:Enables/disables splash window

Type:BOOLEAN

Default:FALSE

initialLayoutWindow

Desc:Enable/disables creation of initial layout window in a new project

Type:BOOLEAN

Default:TRUE

migrateEventHandler

Desc:Enables generation of the #define's that map old event handler names to new ones

Type:BOOLEAN

Default:TRUE

objectScaling

Desc:Turns scaling on (TRUE) or off (FALSE). If scaling is turned off, then Design makes no attempt to scale the windows/controls/dialogs when a Design project file is moved between platforms.

Type:BOOLEAN

showObjectPalette

Desc:Initial visibility of the object palette in each layout window

Type:BOOLEAN

Default:TRUE

showToolbar

Desc:Initial visibility of the toolbar in each layout window

Type:BOOLEAN

Default:TRUE

tabStop

Desc:Sets the tab stops in the ACE

Type:integer

Default:None

XVTCDF

Desc:Path to the CDF directory

Type:string

Default:Platform-specific

XVTDIR

Desc:Path to the top of XVT installation

Type:string

Default:Platform-specific

XVTTPL

Desc:Path to the makefile template directory

Type:string

Default:Platform-specific

7.2.4. Configuration File Example

Here is an example of a configuration file:

```
#Example design.cfg file
#Turn off the splash screen:
initialAboutBox: false
#Set a 16x16 pixel alignment grid:
gridX: 16
gridY: 16
gridNative: false
aceFontFamily: FF_FIXED
aceFontSize: 10
```

7.3. Objects and Tags

This section describes all the tags available for the standard XVT Portability Toolkit user interface objects. To see which tags are available for each object, refer to the table at the end of this section.

7.3.1. Tag Descriptions

This section describes all the tags available in XVT-Design. There are two types of tags:

- **Event tags** correspond to the events in the XVT API: events that are generated by user actions (for example, mouse clicks) or by the window system (for example, update events).
- **Special tags** do not correspond to events but rather serve as convenient places to add functions and variable declarations to the framework code.

7.3.1.1. Event Tags

The following tags correspond directly to XVT Portability Toolkit E_* events. In the Action Code Editor's Tag list button, event tags appear with an "EVNT:" prefix.

See Also: For more detailed descriptions of these events, refer to the *XVT Portability Toolkit Reference.*

Char
The user pressed a key on the keyboard.

Close
The user operated the close control or close menu item on a window or dialog.

Command
Text for this tag is inserted in the window's event handler immediately before the invocation of the window's menubar event handler.

Control
The user operated a control.

Create
A window or dialog has been created. This is the first event that a window or dialog receives; the first event sent to the application is a Create event for the Task window.

Destroy
A window or dialog has been closed. This is the last event received by a window or dialog.

Focus
A window or dialog has gained or lost keyboard focus.

Font
The user has chosen an item from the standard Font menu, or selected a font and style from the font selection dialog.

HScroll
The user has manipulated the horizontal scrollbar on the frame of a document window.

Mouse_Dbl
The mouse button was double-clicked in a window.

Mouse_Down
The mouse button was pressed in a window.

Mouse_Move
The mouse pointer was moved in a window.

Mouse_Up
The mouse button was released in a window.

Quit
The window system is shutting down (not available on all platforms).

Select
The user has selected an item on a menu. Only menu items have the select tag.

Size
The size of a window or dialog has been set or changed.

Timer
The timer associated with a window or dialog has gone off.

Update
The contents of a window require redrawing.

User
An application-defined event has been initiated.

VScroll
The user has manipulated the vertical scrollbar on the frame of a document window.

7.3.1.2. Special Tags

The following tags give you convenient places in the framework code for declaring variables and functions. In the Action Code Editor's Tag list button, special tags appear with an "SPCL:" prefix.

Bottom
Text for this tag is inserted at the end of the object's event-handling function, immediately before a return statement. Use this tag for any "clean-up" you need to perform after the event handler.

Control_Decl
Text for this tag is inserted in the container's event handler, immediately before the code that handles events for controls in the container.

Default
Text for this tag is inserted in the default clause of the switch(tag) statement in the event handler for a menubar.

Help
Text for this tag is placed in your application's help-text file. (By default, this file has a **.csh** extension.)

Caution: The Help tag is provided only for compatibility with existing XVT-Design project files. You should not use it in new projects.

Is_Quit_OK
This tag lets you define an xvt_app_allow_quit function. This function is called in three cases:

- An M_FILE_QUIT menu tag is delivered to the task menu event handler (meaning that the user chose Quit (or Exit) from the File menu).

- An E_CLOSE event is delivered to the application.
- An E_QUIT event is delivered to the application. If this function returns TRUE, then the user's application is terminated.

Main_Code
Text for this tag is inserted in the main function of your application, immediately before calling xvt_app_create.

Obj_Decl (object-wide declarations)
Use this tag for declarations and definitions that are common to all of an object's code (that is, external variables and functions). Text for this tag is inserted at the beginning of the object's source code file, following the #include statements.

User_Header
Text for this tag is placed at the beginning of the header file for your application. Use this tag to declare application-wide type definitions, function prototypes, and so on.

User_URL
Text for this tag is inserted in your application's URL file. The text is placed before the resource statements for your application's resources.

Var_Decl (Variable declarations)
Text for this tag is inserted at the beginning of the object's event-handling function. Define local variables for the event handler here. Any code associated with this tag is executed before the event handler processes each event.

I18N_Header
This tag appears only when Internationalize is selected. Text for this tag is inserted following User_Header. Use this tag to include defines for locale-specific resources. See Chapter 6 of this manual.

I18N_URL
This tag appears only when Internationalize is selected. Text for this tag is inserted following locale-specific resource information. See Chapter 6 of this manual.

Pre_Header
Text for this tag is inserted just before the include for xvt.h. Use this tag to set defines that must be set before platform-specific include files are processed.

I18N_Main
This tag appears only when Internationalize is selected. Text for this tag is inserted following Main-Code. Use this tag for

internationalization-specific code. See Chapter 6 of this manual.

7.3.2. Object/Tag Pairs

The following table describes which tags are available for each type of user-interface object: the application itself, windows, dialogs, controls, menubars, and menu items.

Event Tags	Application	Windows	Dialogs	Controls	Menubars	Menu Items
Char		l	l			
Close	l	l	l			
Command	l	l				
Control				l		
Create	l	l	l			
Destroy	l	l	l			
Focus		l	l			
Font	l	l				
HScroll, VScroll		l				
Mouse Dbl, Mouse Down, Mouse Move, Mouse Up		l				
Quit	l					
Select						l
Size	l	l	l			
Timer	l	l	l			
Update		l				
User	l	l	l			

Special Tags	Application	Windows	Dialogs	Controls	Menubars	Menu Items
Bottom	l	l	l		l	
Control Decl		l	l			
Default					l	
Help	l					
Is Quit OK	l					
Main Code	l					
Obj Decl	l	l	l		l	
User Header	l					
User URL	l					

Var Decl	l	l	l		l	
Pre Header	l	l	l		l	
I18N Main	l					
I18N Header	l					
I18N URL	l					

Note: PreHeader is not I18N-specific. I18N Main, I18N Header, and I18N URL *are* I18N-specific.

7.4. Variables and Constants in Action Code

This section describes the predefined variables and constants that you can use within Action Code.

See Also: For more information about these variable types, see the *XVT Portability Toolkit Reference*.

7.4.1. Variables

The following variables are available in Action Code for window and dialog tags:

WINDOW xdWindow
The WINDOW variable for the module. For control object tags, this variable is the object's container. If you are familiar with XVT Portability Toolkit programming, this variable is traditionally named win in event handlers.

EVENT *xdEvent
The EVENT structure pointer for the module. You can use this structure within Action Code to obtain more detailed information about the event that corresponds to the Action Code's tag. If you are familiar with XVT Portability Toolkit programming, this variable is traditionally named *ep in event handlers.

XVTCM_CONTROL_INFO xdInfo
This variable has information on custom control user events.

7.4.2. Constants

The constants in the following two sections are available in Action Code for window and dialog tags.

7.4.2.1. Dialog Constants

DLG_RES_ID
The resource ID number for the dialog.

DLG_FLAGS
The DLG_FLAG_* option flags that were passed to xvt_dlg_create_def when the dialog was created.

DLG_CLASS
The class name for the dialog. If no class name was specified, this string is empty.

DLG_MODE
The dialog's WIN_TYPE, which is either WD_MODAL or WD_MODELESS.

7.4.2.2. Window Constants

WIN_RES_ID
The resource ID number for the window.

WIN_FLAGS
The WSF_* option flags that were passed to xvt_win_create_def when the window was created.

WIN_CLASS
The class name for the window. If no class name was specified, this string is empty.

WIN_BORDER
The window's WIN_TYPE, which is either W_DOC, W_DBL, W_MODAL, or W_PLAIN.

A

APPENDIX A: THE IMAGE EDITOR

The image editor is an XVT application that lets you create and modify XVT Portability Toolkit portable image files. The image editor has the following features:

- A variety of drawing tools for creating and modifying images
- A text tool for adding text to an image, in any installed font
- A color selector for choosing different foreground, background, and text colors
- A pattern selector for choosing different patterns for the drawing tools
- Several commands for manipulating some or all of the image
- A clipping region, for constraining the area affected by drawing operations

You can use the image editor to create images for use in your XVT-based user interfaces, and for creating preview bitmaps for your custom controls.

A.1. Running the Image Editor

You can either invoke the image editor from within XVT-Design, or by running it as you would any other application.

Tip: To invoke the image editor from within XVT-Design:

Choose Image Editor from the Tools menu.

Tip: To run the image editor from outside of XVT-Design:

Run the **imagedit** application as usual for your development platform, e.g. by double-clicking its icon, or typing **imagedit** on the command line.

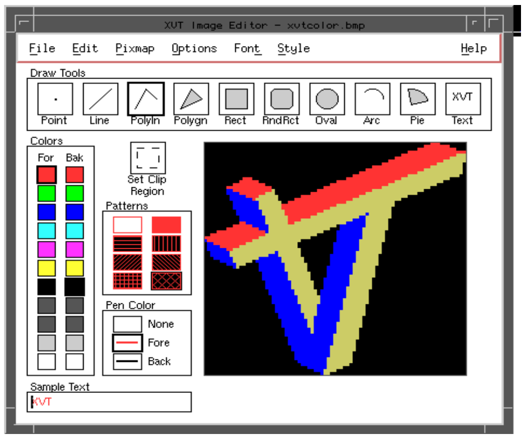

XVT Image Editor - xvtcolor.bmp
File Edit Pixmap Options Font Style Help
Draw Tools
Point Line Polyln Polygn Rect RndRct Oval Arc Pie Text
Colors
For Bak
Set Clip Region
Patterns
Pen Color
None
Fore
Back
Sample Text
XVT

Figure A.1.The Image Editor (Motif Platform)

A.2. Color Selector

The color selector lets you choose the foreground and background colors for the drawing tools.

The *foreground* color is the color that the drawing tools use to draw the patterns within the objects, and text. The *background* color is the color that the drawing tools use to fill the spaces between the the patterns in objects.

The current foreground and background colors are indicated by a border around the corresponding colors in the selector.

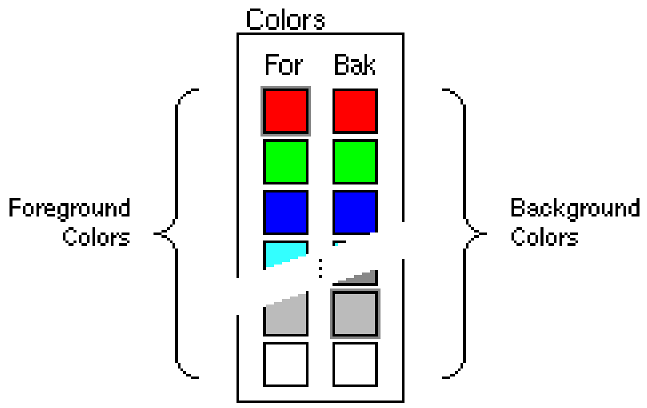


Figure A.2.The Color Selector (Motif Platform)

Tip: To choose the foreground color:

Click the desired color in the column of colors labeled "For" in the color selector.

Tip: To choose the background color:

Click the desired color in the column of colors labeled "Bak" in the color selector.

Note: Choosing a foreground or background color does not affect any existing colors in the image. Only subsequent drawing operations are affected by choosing a new color.

A.3. Pattern Selector

The pattern selector lets you choose the pattern used to fill shapes drawn with the drawing tools.

The patterns in the selector are always drawn in the current foreground and background colors, to show how the patterns will appear in the image.

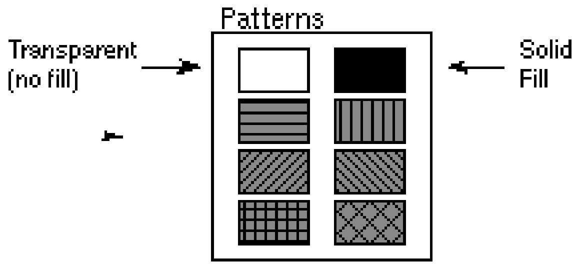


Figure A.3.The Pattern Selector (Motif Platform)

Tip: To choose a pattern:

Click the desired pattern in the pattern selector.

Note: Choosing a pattern color does not immediately affect the image. Only subsequent drawing operations are affected by choosing a new pattern.

A.4. Pen Color Selector

The pen color selector lets you choose the color used to draw single pixels and lines with the drawing tools. The pen color also is used to draw the borders of shapes.

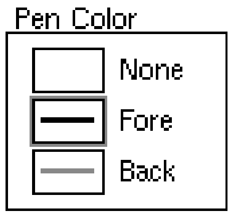


Figure A.4.The Pen Color Selector (Motif Platform)

You can choose either the current foreground or background color

Tip: To set the pen color:

Click the desired color in the pen color selector.

You can choose either the current foreground or background color, or "None". If you choose None, the drawing tools will draw shapes without borders.

Note: Choosing a pen color does not immediately affect the image. Only subsequent drawing operations are affected by changing pen color.

A.5. The Clipping Region

The clipping region is a user-defined rectangular region of pixels. All drawing operations are truncated, or clipped, such that pixels outside of the clipping region are unaffected by the operation.

By default, the clipping region is set to encompass the entire image, so that drawing operations are not clipped (except by the boundaries of the image). The current clipping region is indicated by a rectangle drawn with a thin, dashed line.

Tip: To set the clipping region:

1. Click the Set Clip Region tool icon.
2. Click and hold the mouse button on the pixel where you want to place one corner of the clipping region.
3. While holding the mouse button, drag the pointer. As you move the pointer, a rectangle will stretch from the corner pixel, following the pointer, indicating the size and shape of the clipping region.
4. To finish the clipping region, release the mouse button.

Tip: To remove the clipping region:

Set the clipping region to enclose the entire image, following the steps above
OR
Double-click the Set Clip Region button.

A.6. Drawing Tools

imagedit has a number of drawing tools for creating new images, or modifying existing images. Each tool is described in this section.

A.6.1. Point

Draws single pixels, or a series of pixels, in the current pen color.

Tip: To draw a single pixel:

1. Click the Point tool icon.
2. Click the desired pixel in the image pane.

Tip: To draw a series of pixels:

1. Click the Point tool icon.
2. Click and hold the mouse button on the first desired pixel in the image pane.
3. While holding the mouse button, drag the pointer across more pixels. Each pixel will be colored with the current pen color.
4. Release the mouse button to stop drawing pixels.

A.6.2. Line

Draws a line between two pixels in the current pen color.

Tip: To draw a line:

1. Click the Line tool icon.
2. Click and hold the mouse button on the pixel where you want the line to being.
3. While holding the mouse button, drag the pointer to where you want the line to end. As you move the pointer, a thin line will stretch from the first pixel, following the pointer, indicating where the line will be placed.
4. Release the mouse button to draw the line.

A.6.3. Poly Line (Polyln)

Draws a series of connected line segments in the current pen color.

Tip: To draw a series of line segments:

1. Click the Polyln tool icon.
2. Click the pixel where you want the first line segment to begin.
3. Move the pointer to where you want the segment to end. As you move the pointer, a thin line will stretch from the first pixel, following the pointer, indicating where the line will be placed.
4. Click the pixel where you want the first line segment to end.
5. Move the pointer to where you want the next segment to end, as in step 3.

6. Click the pixel where you want the next segment to end, as in step 4.
7. Repeat steps 5 and 6 to draw additional segments.
8. To finish the last segment, double-click where you want the segment to end.

A.6.4. Polygon (Polygn)

Draws a filled shape with an arbitrary number of sides, such as a triangle or trapezoid. The edges of the shape are drawn in the current pen color, and the shape is filled with the current pattern and foreground/background colors.

Tip: To draw a polygon:

1. Click the Polygn tool icon.
2. Click the pixel where you want to place the first corner of the polygon.
3. Move the pointer to where you want to place the next corner of the polygon. As you move the pointer, a thin line will stretch from the previous corner pixel, following the pointer, indicating where the side of the polygon will be placed.
4. Click the pixel where you want to place the corner of the polygon.
5. Repeat steps 3 and 4 to draw all of the sides of the polygon.
6. To finish the polygon, double-click where you want to place the last corner.

A.6.5. Rectangle (Rect)

Draws a rectangle. The edges of the rectangle are drawn in the current pen color, and the rectangle is filled with the current pattern and foreground/background colors.

Tip: To draw a rectangle:

1. Click the Rect tool icon.
2. Click and hold the mouse button on the pixel where you want to place one corner of the rectangle.
3. While holding the mouse button, drag the pointer. As you move the pointer, a rectangle will stretch from the corner pixel,

following the pointer, indicating the size and shape of the rectangle.

4. To finish the rectangle, release the mouse button.

A.6.6. Rounded Rectangle (RndRct)

Draws a rectangle with rounded corners. The edges of the rectangle are drawn in the current pen color, and the rectangle is filled with the current pattern and foreground/background colors.

Tip: To draw a rounded rectangle:

1. Click the RndRect tool icon.
2. Click and hold the mouse button on the pixel where you want to place one corner of the rectangle.
3. While holding the mouse button, drag the pointer. As you move the pointer, a rectangle will stretch from the corner pixel, following the pointer, indicating the size and shape of the rectangle.
4. To finish the rectangle, release the mouse button.

A.6.7. Oval

Draws an oval. The edges of the oval are drawn in the current pen color, and the oval is filled with the current pattern and foreground/background colors.

Tip: To draw an oval:

1. Click the Oval tool icon.
2. Click and hold the mouse button near where you want to draw the oval. (Imagine the oval placed within a rectangle, and click and hold the mouse on one corner of the imaginary rectangle.)
3. While holding the mouse button, drag the pointer. As you move the pointer, an oval will stretch from the corner pixel, following the pointer, indicating the size and shape of the oval.
4. To finish the oval, release the mouse button.

A.6.8. Arc

Draws a curved line in the current pen color. You draw the arc as a portion of the outline of an oval.

Tip: To draw an arc:

1. Click the Arc tool icon.
2. Click and hold the mouse button near where you want to draw an oval to define the shape of the arc. (Imagine the oval placed within a rectangle, and click and hold the mouse on one corner of the imaginary rectangle.)
3. While holding the mouse button, drag the pointer. As you move the pointer, an oval will stretch from the corner pixel, following the pointer, indicating the size and shape of the oval.
4. Release the mouse button when the oval is the proper size and shape to form the arc.
5. Click and hold the mouse button near the point on the oval where you want the arc to begin.
6. Drag the mouse to indicate the portion of the oval's outline that the arc will cover. As you drag the mouse, the outline of the oval will change to indicate the size and shape of the arc.
   As you draw the arc, you can reverse the direction of drawing (i.e. clockwise or counter-clockwise) by pressing the space bar on your keyboard.
7. To finish the arc, release the mouse button.

A.6.9. Pie

Draws a pie-shaped portion of an oval. The edges of the shape are drawn in the current pen color, and the shape is filled with the current pattern and foreground/background colors.

Tip: To draw a pie-shaped portion of an oval:

1. Click the Pie tool icon.
2. Click and hold the mouse button near where you want to draw the oval. (Imagine the oval placed within a rectangle, and click and hold the mouse on one corner of the imaginary rectangle.)
3. While holding the mouse button, drag the pointer. As you move the pointer, an oval will stretch from the corner pixel, following the pointer, indicating the size and shape of the oval.
4. Release the mouse button when the oval is the proper size and shape.
5. Click and hold the mouse button near the point on the oval where you want the pie to begin.

6. Drag the mouse to indicate the portion of the oval that the pie-shaped portion will cover. As you drag the mouse, the outline of the oval will change to indicate the size and shape of the portion.
   As you draw the arc, you can reverse the direction of drawing (i.e. clockwise or counter-clockwise) by pressing the space bar on your keyboard.
7. To finish the shape, release the mouse button.

A.6.10. Text

Draws text in the current foreground color. Before drawing the text, you must enter the text to be drawn, and choose the font and style for the text.

Tip: To set the text for drawing:

1. Click in the Sample Text edit field.
2. Type the desired text.

Tip: To set the font for drawing text:

1. Choose the desired font from the Font menu(s).
2. Choose the font size and style from the Style menu.

Note: The Font and Style menus vary from platform to platform. On some platforms, they are combined on one menu.

The text in the Sample Text edit field is shown in the selected font and style.

Tip: To draw the text:

1. Click the Text tool icon.
2. Click and hold the mouse button where you want to place the lower-left corner of the text. A thin rectangle shows the boundaries of where the text will be drawn.
3. Drag the mouse to position the boundary rectangle.
4. Release the mouse button to draw the text.

A.7. Menu Commands

This section describes the menu commands available in the image editor.

A.7.1. File Menu

The File menu contains commands for opening and saving image files, creating new image files, and leaving the application.

A.7.1.1. New

Erases the image, so that a new image can be drawn. The size of the image is set to 32x32 pixels, and the name of the previous image is removed from the window's title.

A.7.1.2. Open

Opens a file dialog, so that you can open a previously saved image. In addition to XVT's image format, BMP, XBM, and XPM can also be opened. Macintosh PICT files can be opened with the Macintosh version of **imagedit**.

A.7.1.3. Save

Saves the image. If the image has not been previously saved, this command has the same effect as the Save As command (see below).

A.7.1.4. Save As

Opens a file dialog, so that you can specify a name and location to save the image. The image is stored in the XVT Portability Toolkit portable image format so that you can read it with the xvt_image_read function.

A.7.1.5. Quit

Closes the image editor application.

A.7.2. Edit Menu

A.7.2.1. Undo

Removes the effects of the previously executed command or drawing operation.

A.7.3. Image Menu

The Image menu contains commands for manipulating the image.

A.7.3.1. Change Size

Opens a dialog which allows you to change the size (in pixels) of the image.

Tip: To change the size of the image:

1. Choose Change Size from the Image menu. This opens the Change Image Size dialog.
2. The current size of the image, in pixels, is shown in the Width and Height fields of the Change Image Size dialog. Enter the new width and/or height by editing the the size fields.
3. Click OK to enter the new sizes and dismiss the dialog
   OR
   Click Cancel to dismiss the dialog without changing the size of the image.

A.7.3.2. Crop to Clip Region

Discards all pixels outside of the current clipping region, and sets the size of the image to equal the size of the clipping region.

A.7.3.3. Clear

Sets the color of all pixels inside the clipping region to black.

A.7.3.4. Flip Horizontal

Creates a mirror image of the clipping region by flipping the region around its vertical axis. Pixels on the right of the region are moved to the left, and vice versa. (See illustration below.)

A.7.3.5. Flip Vertical

Flips the clipping region around its horizontal axis. Pixels on the top of the region are moved to the bottom, and vice versa. (See illustration below.)

A.7.3.6. Rotate

Rotates the clipping region and the pixels it encompasses 90 degrees clockwise.

The following illustration shows the effects of the Flip Horizontal, Flip Vertical, and Rotate commands:

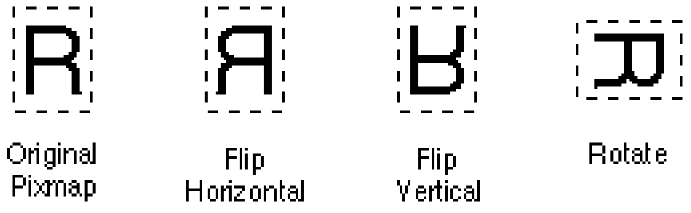


Figure A.5.Flip and Rotate Commands

A.7.3.7. Shift Left

Moves the clipping region, and the pixels it contains, one pixel to the left.

A.7.3.8. Shift Right

Moves the clipping region, and the pixels it contains, one pixel to the right.

A.7.3.9. Shift Up

Moves the clipping region, and the pixels it contains, one pixel towards the top of the image.

A.7.3.10. Shift Down

Moves the clipping region, and the pixels it contains, one pixel towards the bottom of the image.

A.7.4. Options Menu

The Options menu contains commands that affect how the image is displayed in the editor.

A.7.4.1. Show Drawing Grid

Draws a grid of thin lines in the image editing pane, indicating the divisions between pixels.

If the grid is already shown, choosing this command removes the grid. If the grid is currently shown, a check mark appears next to this menu item.

A.7.4.2. Show Image Window

Opens a small window which displays the image in its actual size and color. All changes you make to the image are also shown in this window. You can move this window to any convenient location, by clicking and dragging in the content area of the window.

If the image window is already open, choosing this command closes the window. If the window is open, a check mark appears next to this menu item.

A.7.5. Font and Style Menus

Sets the font, font size, and style that the Text tool uses to draw text. The Sample Text edit field draws the text according to current font settings.

On some systems (for example, MS-Windows) the Font and Style menus are combined into a single Font menu.

XVTTPL 16