XVT Platform-Specific

Guide-Mac

XVT/MAC

CONTENTS

Preface 1-ix
About This Manual 1-ix
Conventions Used in This Manual 1-ix

Chapter 1: Introduction 1-1
1.1. Compilers Supported by XVT/Mac 1-1
1.2. XVT Implementations and Operating Systems 1-2

Chapter 2: Using XVT/Mac 2-1
2.1. Introduction 2-1
2.1.1. Native Look-and-feel Development Issues 2-1
2.1.2. Sample Porting Restrictions — XVT/Mac 2-1
2.1.2.1. Multiple Monitors 1
2.1.2.2. Titlebars versus Menubars 2
2.2. Extensibility 2-2
2.2.1. Conditional Compilation 2-2
2.2.2. Accessing Window Device Contexts and Handles 2-3
2.2.3. Macintosh Toolbox Initialization 2-4
2.2.4. Accessing the Macintosh Toolbox File Manager 2-4
2.3. Invoking an Input Method Editor 2-5
2.4. XVT/Mac Resource Specifics 2-5
2.4.1. Bundle Resources 2-6
2.4.2. SIZE Resources 2-9
2.4.3. Version Resources 2-11
2.4.4. Window Resources 2-12
2.4.5. Dialog Resources 2-12
2.4.5.1. Modal Dialogs 13
2.4.5.2. Movable Modal Dialogs 15
2.4.5.3. Automatic Dialog Positioning 16

2.4.5.4. Color Dialogs 17
2.4.6. Control Resources 2-19
2.4.6.1. Control “Ground” Rules 19
2.4.6.2. Rez Definition of the CNTL Resource 20
2.4.7. Menu Resources 2-23
2.4.7.1. Creating Macintosh-specific Menus 23
2.4.7.2. Menu Item Numbering 24
2.4.7.3. “Quit” Menu Item 24
2.4.7.4. Color Menus 24
2.4.7.5. Menu Accelerators 26
2.4.7.6. Balloon Help Menu Access 26
2.4.8. Cursor Resources 2-29
2.4.9. Control Icon Resources 2-29
2.4.10. Drawn Icons Resources 2-31
2.4.11. Finder Icon Resources 2-31
2.4.12. String Resources 2-33
2.4.12.1. International Strings 33
2.4.12.2. String Resource IDs 34
2.5. XVT’s Encapsulated Font Model 2-35
2.5.1. Font Terminology 2-35
2.5.2. Native Font Descriptors 2-35
2.5.2.1. XVT/Mac Font Descriptor Version Identifier 35
2.5.2.2. XVT/Mac Font Fields 36

Chapter 3: Development Environment 3-1

3.1. Introduction 3-1
3.1.1. Include Files 3-2
3.2. Metrowerks CodeWarrior C/C++ Development Environment 3-3
3.2.1. Environment Options: Metrowerks CodeWarrior C/C++ 3-3
3.2.2. Link Libraries 3-5
3.2.3. For Source Customers Only: XVT/Mac Development Environment 3-6
3.2.3.1. Building the XVT/Mac Libraries 6
3.2.3.2. Building Utility Programs 6
3.3. Compiling Resources 3-8
3.3.1. Using the xrc Interactive Interface 3-8
3.3.1.1. Using Drag and Drop with xrc 9
3.3.2. Macintosh Resource Compilers 3-9
3.3.2.1. Metrowerks CodeWarrior Rez Compiler 9

3.4. Building Your Application with the Help System 3-10
3.4.1. Portable Viewers..... 3-10
3.4.2. Using the Helpc Interactive Interface 3-10
3.4.2.1. Bound Viewer..... 11
3.4.2.2. Standalone Viewer 12

Appendix A:
Non-portable Attributes and
Escape Codes..... A-1

A.1. Non-portable Attributes..... A-1
ATTR_MAC_ALWAYS_UPDATE A-1
ATTR_MAC_BEHIND_WINDOW A-2
ATTR_MAC_CHAR_TO_TASK A-2
ATTR_MAC_CONTROL_HANDLE A-3
ATTR_MAC_CTL_DEFER_UPDATE A-3
ATTR_MAC_EVENT_TIME A-4
ATTR_MAC_FOREIGN_WIN A-4
ATTR_MAC_FRONT_WINDOW_FCN A-4
ATTR_MAC_HAVE_COLOR_QUICKDRAW A-5
ATTR_MAC_HILITE_MODE A-5
ATTR_MAC_LBOX_KEY_HOOK A-6
ATTR_MAC_LBOX_PROC_ID A-7
ATTR_MAC_LOW_MEMORY_THRESHOLD A-7
ATTR_MAC_MENU_HOOK A-8
ATTR_MAC_MOUSE_CONTROL_FOCUS A-9
ATTR_MAC_NATIVE_HTML_REFERENCE A-9
ATTR_MAC_NO_GRAY_DISABLED_EDIT A-10
ATTR_MAC_NO_GRAY_MAP_COLORS A-10
ATTR_MAC_NO_LBOX_FOCUS_BOX A-11
ATTR_MAC_NO_SELECT_WINDOW A-11
ATTR_MAC_NO_SET_CURSOR A-12
ATTR_MAC_NO_UPDATE_MENU_BAR A-12
ATTR_MAC_PAT_RES_ID A-13
ATTR_MAC_PAT_RES_INDEX A-13
ATTR_MAC_PIXMAP_GWORLD_DEPTH A-14
ATTR_MAC_PRINT_CLIPPING A-14
ATTR_MAC_PRINT_COPIES A-15
ATTR_MAC_PRINT_FIRST_PAGE A-15
ATTR_MAC_PRINT_LAST_PAGE A-16
ATTR_MAC_PROC_ID A-16
ATTR_MAC_ROUNDED_GROUPBOX A-17

ATTR_MAC_SCROLL_THUMBTRACK A-17
ATTR_MAC_SET_TITLE_AUTO_SELECT A-18
ATTR_MAC_SHOW_JOB_DIALOG A-18
ATTR_MAC_STR_HELP A-19
ATTR_MAC_STR_STYLE_MENU1 ..... A-20
ATTR_MAC_STR_STYLE_MENU2 ..... A-20
ATTR_MAC_STR_STYLE_MENU3 ..... A-21
ATTR_MAC_SYSTEM_INITIALIZATION A-22
ATTR_MAC_USE_COLOR_QUICKDRAW A-23
ATTR_MAC_USE_NATIVE_ORIGIN A-23
ATTR_MAC_WIN_MAX_HEIGHT
ATTR_MAC_WIN_MAX_WIDTH
ATTR_MAC_WIN_MIN_HEIGHT
ATTR_MAC_WIN_MIN_WIDTH A-24
ATTR_MAC_WIN_USE_FIRST_CLICK A-24
A.2. Variations on Portable Attributes A-25
ATTR_EVENT_HOOK A-25
ATTR_HAVE_COLOR A-26
ATTR_HAVE_MOUSE A-26
ATTR_KEY_HOOK A-27
ATTR_NATIVE_GRAPHIC_CONTEXT A-29
ATTR_NATIVE_WINDOW A-30
ATTR_NUM_TIMERS A-30
ATTR_PRINTER_* A-31
A.3. Non-Portable Escape Codes A-31
XVT_ESC_MAC_DIALOG_POSITION A-31
XVT_ESC_MAC_FONT_GET_RES_NAME A-32
XVT_ESC_MAC_GET_DESKTOP_BOUNDS A-32
XVT_ESC_MAC_GET_DISPLAY_INFO A-32
XVT_ESC_MAC_GET_EDIT_HANDLE A-33
XVT_ESC_MAC_GET_LIST_HANDLE A-33
XVT_ESC_MAC_GET_PICT_ID A-34
XVT_ESC_MAC_MODAL_WINDOW A-34
XVT_ESC_MAC_PALET_GET_PALETTE_HANDLE A-35
XVT_ESC_MAC_PICT_READ_FROM_FILE A-35
XVT_ESC_MAC_PICTURE_COMMENT A-35
XVT_ESC_MAC_RES_GET_PICT A-36
XVT_ESC_MAC_SET_PICT_ID A-36
XVT_ESC_MAC_SET_WINDOW_COLOR A-37

Appendix B:

Frequently Asked Questions.....B-1
Index.....I-1

XVT/MAC

PREFACE

About This Manual

XVT takes pride in its documentation, and continually seeks to improve it. If you find a documentation error, please contact Customer Support. They will forward your suggestion to XVT's documentation team.

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.

code bold
: This typestyle is used for elements that you see in the user interface of applications, such as compilers and debuggers. An arrow separates each successive level of selection that you need to make through a series of menus, e.g., **Edit=>Font=>Size**.

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

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

Tip: This marks the beginning of a procedure having one or more steps. Tips can help you quickly locate “how-to” information..

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

This symbol and typestyle highlight information specific to using XVT-Design, XVT’s C visual programming tool and code generator.

This symbol and typestyle highlight information specific to using XVT-Architect, XVT’s C++ visual programming tool and code generator.

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.

...
: Ellipses in data values and data types indicate that these values and types are opaque. You should *not* depend upon the actual values and data types that may be defined.

1

INTRODUCTION

Welcome to XVT/Mac. This platform-specific book (PSB) contains information about using the latest release of the XVT Portability Toolkit (XVT/Mac) on your particular platform. If you had an earlier version of XVT/Mac, this manual replaces the previous platform-specific book.

Your release media will have instructions for installing XVT/Mac. Once you have XVT/Mac installed, XVT recommends that you read this book and try the sample programs that come with the product.

Note: Before writing your application, read the *XVT Portability Toolkit Guide*. The *Guide* focuses on strategies for developing portable applications.

See Also: For an alphabetical listing of all XVT functions and other API elements, refer to the *XVT Portability Toolkit Reference*.
For additional information not documented in this platform-specific book, see the **readme** file in the **doc** folder.

1.1. Compilers Supported by XVT/Mac

XVT/Mac supports one compiler, Metrowerks CodeWarrior C/C++.

See Also: Changes to compiler support may be listed in the **readme** file in the **doc** folder.

1.2. XVT Implementations and Operating Systems

The XVT library is currently available for several different window systems and operating systems:

XVT Product:	Window Systems:	Operating Systems:
XVT/Mac	Carbon	MacOS 8.6 and above with CarbonLib installed, Mac OS X 10.1 and above
XVT/Win32	Win32	Windows NT, XP, Me, Windows 95, 98, 2000
XVT/XM	X and Motif	UNIX

2

USING XVT/MAC

2.1. Introduction

This chapter addresses various platform-specific issues that you may need to consider while using XVT/Mac. The information here assumes you are familiar with developing Macintosh applications from a general standpoint. If not, see *Inside Macintosh* for more information.

2.1.1. Native Look-and-feel Development Issues

Developing successful cross-platform applications demands familiarity with the look-and-feel of each target platform. This requirement is especially true when developing for the Macintosh. The next section outlines some sample porting restrictions for XVT/Mac applications.

See Also: For more information on native look-and-feel differences, refer to the other platform-specific books you may have received.

2.1.2. Sample Porting Restrictions — XVT/Mac

Applications ported to XVT/Mac need to take into account Macintosh screen sizes and interface look-and-feel.

2.1.2.1. Multiple Monitors

XVT/Mac automatically handles multiple monitors, allowing your end users to: 1) drag windows from one monitor to the other, and 2) resize windows so they span both monitors. If cross-platform portability is important, your application should not make *explicit* use of multiple monitor information, because XVT/Mac supports this behavior automatically. On the other hand, if it seems necessary,

and if you don't mind sacrificing portability, your application can access (non-portably) layout information of multiple monitors using platform-specific escape functions.

The platform-specific escape function XVT_ESC_MAC_GET_DISPLAY_INFO returns the bounding rectangle of a specified monitor, and the new platform-specific escape function XVT_ESC_MAC_GET_DESKTOP_BOUNDS returns the bounding rectangle of the union of the monitor rectangles. For more details, see the descriptions of these two escape functions in Appendix A.

2.1.2.2. Titlebars versus Menubars

In general, maintaining consistency in application ports is desirable. For example, you might want to maintain the relative placement of menubars and titlebars between platforms. However, this is not always possible when retaining native look-and-feel. For example, on XVT/Mac, the title of the application doesn't hold the visually dominant position relative to the menubar that it does on other platforms.

2.2. Extensibility

2.2.1. Conditional Compilation

If, in your application, you need to provide some native-platform GUI functionality not available in the XVT Portability Toolkit, then the small percentage of your code that provides that functionality will be non-portable. In this case, you must compile your code conditionally, based on the compilers, the window systems, file systems, and the operating system on which your non-portable code will run.

The XVT Portability Toolkit automatically determines the environment in which the application is running. The XVT header files automatically sense the environment-specific variables that are set implicitly by the compiler.

See Also: For more information on conditional compilation, see the "Symbols for Conditional Compilation" section in the "About the XVT API" chapter of the *XVT Portability Toolkit Guide*, and the file **xvt_env.h** in your **include** folder.

Tip: It's best to consolidate the non-portable code into a few separate files so that most of your application will be portable XVT code. Separating your non-portable code makes it easier to change your

program when the capability you need is added to a future version of XVT.

Tip: To compile Macintosh-specific code conditionally:

Use the following preprocessor statements to compile window-system-specific code:

```
#if XVTWS == MACWS
/* window-system-specific code goes here */
#endif
```

Use the following preprocessor statements to compile file system-specific code:

```
#if XVT_FILESYS_MAC
/* file-system-specific code goes here */
#endif
```

2.2.2. Accessing Window Device Contexts and Handles

Given an XVT WINDOW, your application can access its native window handle (WindowRef) or graphics context (CGrafPtr). Color QuickDraw is now supported under all systems. As a result, the attribute ATTR_MAC_HAVE_COLOR_QUICKDRAW (to determine if Color QuickDraw is available) is obsolete. Any reference to non-Carbon compliant WindowPtr or CWindowPtr will need to be changed to WindowRef. References to GrafPtr and CGrafPtr are interchangable and should be standardized to CGrafPtr.

Tip: To get the Macintosh WindowRef associated with an XVT WINDOW (excluding windows of type W_PIXMAP, W_PRINT and W_SCREEN) use logic similar to the following:

```
(WindowRef) xvt_vobj_get_attr(win, ATTR_NATIVE_WINDOW);
```

Tip: To get the Macintosh CGrafPtr associated with an XVT WINDOW (includes only drawable windows of type W_DOC, XVT Platform-Specific Book for Macintosh W_PLAIN, W_DBL, W_MODAL, W_TASK if drawable, and W_NO_BORDER) use logic similar to the following:

```
(CGrafPtr) xvt_vobj_get_attr(win, ATTR_NATIVE_GRAPHIC_CONTEXT);
```

See Also: For complete descriptions of these attributes, see Appendix A.

2.2.3. Macintosh Toolbox Initialization

Using XVT-Design, you can initialize certain Macintosh Toolbox systems by editing the tag SPCL:Main_Code *in the Action Code Editor (ACE) and setting the attribute* ATTR_MAC_SYSTEM_INITIALIZATION.

XVT/Mac initializes certain Macintosh Toolbox systems with an internal function contained in the file **minit.c** in the **samples:ptk:mac** folder. To initialize the Macintosh Toolbox before calling xvt_app_create, your application can create its own initialization function and assign it to the attribute ATTR_MAC_SYSTEM_INITIALIZATION. The function xvt_app_create automatically calls the function assigned to the attribute ATTR_MAC_SYSTEM_INITIALIZATION instead of its internal function. Your application can call your initialization function directly prior to calling xvt_app_create. However, you must ensure that you protect your function against multiple calls. Note the use of the static variable mac_initialized in **minit.c**.

If your application needs to call other Macintosh Toolbox initialization functions, you can copy the file **minit.c** and create your own function by making additions to the internal function in **minit.c**. However, do not remove anything from this function, because XVT/Mac depends on the Toolbox initialization performed in this function.

Using XVT-Architect, you can initialize certain Macintosh Toolbox systems by editing the application's StartUp *method before calling the base class,* CApplication::StartUp *method and setting the attribute, ATTR_MAC_SYSTEM_INITIALIZATION.*

2.2.4. Accessing the Macintosh Toolbox File Manager

The FILE_SPEC record contains a field of type DIRECTORY. In XVT/Mac this type is defined as containing the Mac volume ID and the directory ID, as follows:

```
typedef struct s_dir {
        short vol;
        long dir;
} DIRECTORY;
```

This gives the application access to the native Mac volume ID and directory ID by referencing the fields dir.vol and dir.dir in the

FILE_SPEC record to use with the Macintosh Toolbox File Manager functions.

2.3. Invoking an Input Method Editor

An Input Method Editor (IME) is provided by Apple with the operating system or language kit to allow application users to enter multibyte or other non-ASCII characters from a keyboard that does not support these characters. On the Mac, users may select a script system on the keyboard layout menu. The IME appears automatically as a window at the bottom of the screen as the user types. The IME window may be moved, sized, and closed by the user. In IMEs that use composed characters (for example in the conversion from Katakana to Kanji), double clicking on the word or pressing the space bar performs the composition of selected characters. The IME is disabled when the user is finished entering characters (Enter or Return key).

Character events are sent at appropriate times as determined by the IME. If the user composes a character, a character event is sent only *after* the conversion—only the composed character is sent in the event. This means that there may be a delay between when characters are typed and when your application receives an event. Several characters may be typed before any character event is received.

2.4. XVT/Mac Resource Specifics

You will probably never need to code native resources directly, since XVT-Design and ***xrc*** *handle the job automatically when you use the XVT-Design tag* SPCL:User_URL*. The following information is provided for reference only.*

Generating Macintosh-specific Resources in XVT-Design

Tip: To generate platform-specific resources in XVT-Design:

1. Select the tag SPCL:User_URL in the ACE.
2. Add Mac conditionally-compiled Rez statements with an URL #transparent statement.

XVT-Design's code generation facility adds the code fragment to the generated URL file after **url.h** and after the XVT-Design-specific URL statements.

See Also: For more information, see the "Objects and Tags" section in the "Reference" chapter of the *XVT-Design Manual*.

Coding Macintosh-specific Resources

This section tells you how to code Macintosh-specific resources that can be used with XVT. If you plan to code resources exclusively in URL, XVT's Universal Resource Language, you don't need to read this section, except as background information. Before creating any Macintosh-specific resources, read the "Resources and URL" chapter in the *XVT Portability Toolkit Guide*.

You can code all menus, dialogs, and strings in URL. Or, you can create them directly in binary resources with a resource editing tool such as **ResEdit**. Because XVT/Mac makes special use of some native Macintosh resource types, coding resources by means other than URL is not recommended. If you must code them in Rez, study the output of **xrc** (in Rez format) for the resources in the XVT Example Set to see how they are coded.

When defining macros that must be invoked prior to including **url.h**, you may either define the macro in your source header files or on the **xrc** compile line or in the options file **xrc.opt**.

2.4.1. Bundle Resources

Your Macintosh application should include a bundle resource that specifies its version number, creator ID, application icon, and types and icons for each of the files that it can process.

See Also: For more information about bundle resources, see the "Finder Interface" chapter in the book *Inside Macintosh: Macintosh Toolbox Essentials*.

When you code in URL, **url.h** automatically includes a default bundle resource. If you want to code your own bundle resource, you must define the symbol NO_STD_BUNDLE before including **url.h**, then define your own bundle resource with an URL #transparent statement.

Example: This example show how bundle resources are used in an application named XVT-Sample:

```
#if XVTWS == MACWS
#transparent $$$
                        /* Signature resource */
type 'XVTD' as 'STR ';                          /* XVTD is signature */
resource 'XVTD' (0) {                           /* Creator resource
                                                    ID must be 0 */
        "XVT-Sample Version M1.2R"              /* Finder Info string */
};

                        /* File reference resource */
resource 'FREF' (128) {                         /* File type reference for
                                                    application icon */
        'APPL',                                 /* File type */
        0,                                      /* Local icon ID is 0 */
        ""                                      /* String must be empty */
};
                        /* Sample icon file type reference */
resource 'FREF' (129) {
        'VTD ',                                 /* File type */
        1,                                      /* Local icon ID is 1 */
        ""                                      /* This string should
                                                    be empty, unused */
};
                        /* Help icon file type reference */
resource 'FREF' (130) {
        'XVTH',                                 /* File type */
        2,                                      /* Local icon ID is 2 */
        ""                                      /* String must be empty */
};
                        /* Bundle resource */
resource 'BNDL' (128) {
        'XVTD',                         /* Signature of this application */
        0,                              /* Creator resource ID (must be 0) */
        {
                'ICN#',                 /* Local resource icon ID mapping */
                {
                        0, 128,         /* ICN# local ID 0 maps to 128 */
                        1, 129,         /* ICN# local ID 1 maps to 129 */
                        2, 130          /* ICN# local ID 2 maps to 130 */
                },
                'FREF',                 /* Local resource ID file mapping*/
                {                       /* type references */
                        0, 128          /* FREF local ID 0 maps to 128 */
                        1, 129,         /* FREF local ID 1 maps to 129 */
                        2, 130          /* FREF local ID 2 maps to 130 */
                }
        }
};
$$$
#endif
```

The following description of XVT-Sample illustrates the various parts of a bundle:

- The creator is XVTD. A string resource of type XVTD, called the signature, notes the version number. The signature is displayed from the Finder when the “Get Info” command is issued. The creator also appears in the first line of the BNDL resource.
- XVT-Sample has three icons and two data-file types. The first icon (number 128) is for the program itself. It is

associated via FREF 128 with the file type APPL, which is the type for all Macintosh applications. XVT-Sample also has an icon and file type (VTD) for its own application-related files.

The VTD file reference type isn't a standard type—it was invented for XVT-Sample. Types you invent should have only three characters for portability, because DOS-based systems limit types to three characters. However, because the Macintosh uses four-character file types, you must append a space. This is necessary only when you edit Rez text files. It is not necessary when you use xvt_fsys_set_file_attr with XVT_FILE_ATTR_TYPESTR.

- XVT fixes the last type, XVTH, as the type for compiled help files. XVT-Sample lists it in the BNDL so that a special icon can exist for it. (You don't have to list XVTH in your bundle in order to use XVT's online help system in your application.)
- Not shown in the Rez input are the three icon resources (128, 129, and 130). These were put into the resource file with **ResEdit**. They could as easily have been put into their own resource file and referenced by a Rez include statement. This include statement would itself have to be in an URL #transparent statement, since an URL #include statement cannot be used to include binary resources.

You must make sure you set your application's creator to match what you've used in the bundle resource. For information on how to set your application's creator, please refer to the compiler's reference manual.

Caution: It is a common mistake to specify *conflicting* application information in the Set Project Type dialog and URL-based SIZE resource—this results in a runtime error. Make sure that correct information is specified in both places.

If your files' icons do not appear in the Finder, use **ResEdit** to verify that they are in the application file and that the creator and type of the application file are correct. Then use **ResEdit** to clear the Desktop file, and reboot the Macintosh. When you reboot, the Finder reconstructs the Desktop file.

2.4.2. SIZE Resources

If you simply need to set the partition sizes, you do not have to create your own SIZE resource.

Tip: To set partition sizes, use the **xrc** compile line (or **xrc.opt** file) -d option:

```
-d XVT_MINIMUM_MEM_SIZE = "800*1024"
-d XVT_PREFERRED_MEM_SIZE = "2048*1024"
```

The Macintosh SIZE resource communicates Finder attribute information about your application to the Finder. It also specifies the minimum and preferred partition sizes for your application:

- The minimum partition size is the actual limit below which your application cannot run.
- The preferred partition size is the memory size at which your application runs most effectively and the amount of memory the operating system attempts to secure upon launching your application. Your application must contain a SIZE resource to be Carbon compliant.

When you code in URL, **url.h** automatically includes a default SIZE resource. The default partition sizes are set to (2048 * 1024) or 2 MB for the minimum size and (4096 * 1024) or 4 MB for the preferred size.

Tip: To specify your own SIZE resource:

1. Define the symbol NO_STD_SIZE before including **url.h**.
2. Define the bundle resource with an URL #transparent statement.

Metrowerks CodeWarrior C++ determines the SIZE resource from the settings for the project. Select “Preferences” on the Edit menu to invoke the preferences dialog. Select Project from the scroll list.

Implementation Note: For applications developed with XVT/Mac for Power Macintosh, the field AcceptHighLevelEvents *must* be set to isHighLevelEventAware. By default, this value is set to true
for the Power Macintosh platform. A runtime error occurs if this flag is not set properly.

The following sample code uses URL to set the minimum partition size to 4 MB and the preferred partition size to 8 MB:

```
#define XVT_MINIMUM_MEM_SIZE 4*1024*1024
#define XVT_PREFERRED_MEM_SIZE 8*1024*1024
#include “url.h”
```

Example: The following example (found in the file **url_plat.h** in the **include** folder) shows the XVT/Mac default SIZE resource, including the attribute value settings that XVT uses.

```
#ifndef NO_STD_SIZE
#transparent $$$
resource 'SIZE' (-1) {                  /* Default SIZE resource for XVT app */
        reserved,                       /*      Reserved */
        acceptSuspendResumeEvents,      /* Suspend/resume event-aware - */
                                        /*          ignoreSuspendResumeEvents, */
                                        /*          acceptSuspendResumeEvents */
        reserved,                       /*      Reserved */
        canBackground,                  /* Can properly use background null events - */
                                        /*          cannotBackground, */
                                        /*          canBackground */
        doesActivateOnFGSwitch,         /* Activate/deactivate on resume/suspend - */
                                        /*          needsActivateOnFGSwitch, */
                                        /*          doesActivateOnFGSwitch */
        backgroundAndForeground,        /* Application does not have a user interface - */
                                        /*          backgroundAndForeground, */
                                        /*          onlyBackground */
        dontGetFrontClicks,             /* Get mouse down/up when suspended - */
                                        /*          dontGetFrontClicks, */
                                        /*          getFrontClicks */
        ignoreAppDiedEvents,            /* Apps use ignoreAppDiedEvents */
                                        /* Debuggers use acceptAppDiedEvents - */
                                        /*          ignoreAppDiedEvents, */
                                        /*          acceptAppDiedEvents */
        is32BitCompatible,              /* Works with 24-bit addresses only or works */
                                        /* with 24- or 32-bit addresses - */
                                        /*          not32BitCompatible, */
                                        /*          is32BitCompatible */

                /* Next four bits are for System 7.0 */
        isHighLevelEventAware,          /* Posts/accepts high level events */
                                        /*          notHighLevelEventAware, */
                                        /*          isHighLevelEventAware-Use for PowerPC */
        onlyLocalHLEvents,              /* Local or local/remote high level events - */
                                        /*          onlyLocalHLEvents, */
                                        /*          localAndRemoteHLEvents */
        notStationeryAware,             /* Checks stationery bit when opening documents - */
                                        /*          notStationeryAware, */
                                        /*          isStationeryAware */
        dontUseTextEditServices         /* Can use text services through TextEdit - */
                                        /*          dontUseTextEditServices, */
                                        /*          useTextEditServices */
        reserved,                       /* Reserved */
        reserved,                       /* Reserved */
        reserved,                       /* Reserved */

                /* Memory sizes are in bytes */
#ifdef XVT_PREFERRED_MEM_SIZE
        XVT_PREFERRED_MEM_SIZE,
#else
        4 * 1024 * 1024,                /* Preferred memory size */
#endif
```

```
#ifdef XVT_MINIMUM_MEM_SIZE
        XVT_MINIMUM_MEM_SIZE,
#else
        2 * 1024 * 1024                 /* Minimum memory size */
#endif
};

$$$
#endif
```

See Also: For more detailed information on SIZE resources, see the "Event Manager" chapter in *Inside Macintosh: Macintosh Toolbox Essentials*.

2.4.3. Version Resources

The Macintosh version resource stores your application version information for use by the Finder. This information is displayed when the user selects **Get Info** from the **File** menu.The Finder also uses the version resource to display version information if the user opens the **Views** control panel and selects the **Show Version** box.

The vers resource stores a version number, a version message, and a region code. The vers resource Rez template is as follows:

```
type 'vers' {
        byte;           /* Major version number */
        byte;           /* Minor version number */
        byte;           /* Development, alpha, beta, release */
        byte;           /* Stage of prerelease version */
        integer;        /* Region code, as in Script
                           Manager */
        pstring;        /* Version number string */
        pstring;        /* Version message string */
}
```

The version message does not need to contain the application name. The application name comes from the file name in the Finder.

Example: This example defines a version resource for **xrc.app**:

```
#if XVTWS == MACWS
#transparent $$$

resource 'vers' (1, purgeable) {
        0x05,
        0x50,
        release,
        0x00,
        verUS,
        "5.60",
        "5.60, © Providence Software Solutions, Inc. 1989-2002"
};

$$$
#endif
```

2.4.4. Window Resources

XVT/Mac does not support the native Mac WIND resource. To create windows from a resource, see the "Window Statement" topic in the URL section of the online *XVT Portability Toolkit Reference*.

2.4.5. Dialog Resources

The following subsections describe rules you must follow when creating dialogs for use with XVT/Mac. If you're coding in URL, you don't have to follow these rules or know how to code dialogs in Rez. Just follow the rules in the "Resources and URL" chapter in the *XVT Portability Toolkit Guide* and the URL section of the online *XVT Portability Toolkit Reference*.

Macintosh dialog resources are of type DLOG, and the corresponding resource ID is the second argument of the xvt_dlg_create_res function. The DLOG resource references a DITL resource, which lists the controls. You can use the following controls:

check box	list box	push button
edit	list button	radio button
group box	list edit	scrollbar
icon	picture	static text

Note: XVT does not support items of type userItem in dialogs, even when coded in native resource format.

Note: If you're using an interactive utility such as **ResEdit** to create resources, you'll need to familiarize yourself with XVT dialog resources, because mode controls, such as list boxes, are handled and ordered in a special way. If in doubt, use **ResEdit** to examine the resource fork you've created.

2.4.5.1. Modal Dialogs

To see how to code for a modal dialog, look at the About box example below. Note that the resource controls only the *appearance* of the box. Whether it *behaves* modally is determined by the first argument (WD_MODAL or WD_MODELESS) to xvt_dlg_create_res or xvt_dlg_create_def. Since XVT About boxes are always modal, provide a resource that matches this behavior.

Rez Definition of the DLOG Resource

XVT/Mac stores information in certain fields in the dialog resource record.

Example: This example shows the Rez definition of the DLOG resource:

```
type 'DLOG' {
        rect;                                   /* boundsRect */
        integer documentProc,                   /* procID */
                dBoxProc,
                plainDBox,
                altDBoxProc,
                noGrowDocProc,
                movableDBoxProc,
                zoomDocProc = 8,
                zoomNoGrow = 12,
                rDocProc = 16;
        byte invisible, visible;                /* visible */
        fill byte;
        byte noGoAway, goAway;                  /* goAway */
        fill byte;
        unsigned hex longint;                   /* refCon */
        integer;                                /* 'DITL' ID */
        pstring;                                /* title */
/* The following are options for positioning windows,
        usable in 7.0 */
#if SystemSevenOrLater
        align word;
                unsigned integer noAutoCenter = 0x0000,
                centerMainScreen = 0x280a,
                alertPositionMainScreen = 0x300a,
                staggerMainScreen = 0x380a,
                centerParentWindow = 0xa80a,
                alertPositionParentWindow = 0xb00a,
                staggerParentWindow = 0xb80a,
                centerParentWindowScreen = 0x680a,
                alertPositionParentWindowScreen = 0x700a,
                staggerParentWindowScreen = 0x780a;
#endif
};
```

DLOG Resource Definition Fields

You must set the procID field to documentProc for modeless dialogs or dBoxProc for modal dialogs.

The refCon field of the DLOG resource passes XVT dialog flags from URL to XVT/Mac, since no native fields support these features. The available flags are:

MAC_FLAG_DISABLED	Initially disables the dialog
MAC_FLAG_MODAL	Specifies that the dialog is modal

Rez Definition of the DITL Resource

All XVT/Mac controls are defined within the DITL resource as Control dialog item types that specify the resource ID of a CNTL resource.

Tip: To create XVT controls:

1. Specify all dialog items as type Control.
2. Enter a CTRL resource ID in the item.

Example: This code fragment shows an extraction of the Rez definition of the DITL resource:

```
type 'DITL' {
        integer = $$CountOf(DITLarray)-1;          /* Array size */
        wide array DITLarray {
                fill long;
                rect;                              /* boundsRect */
                switch {
                case Control:
                        boolean enabled,
                                disabled;          /* Enable flag */
                        key bitstring[7] = 7;      /* ctrlItem and
                                                      resCtrl */
                        byte = 2;
                        integer;                   /* 'CTRL' ID */
                };
                align word;
        };
};
```

2.4.5.2. Movable Modal Dialogs

You can create Macintosh movable modal dialogs with a call to an xvt_win_create_* function using the W_MODAL type. Alternatively, you can create XVT modal dialogs as Macintosh movable modal dialogs by using the attribute ATTR_MAC_PROC_ID.

See Also: For more information on how to create portable modal windows (W_MODAL type), see the "Modal Windows" section in the "Windows" chapter of the *XVT Portability Toolkit Guide*.

Tip: To create Macintosh movable modal dialogs:

1. Before creating your modal dialog, set the value of the attribute ATTR_MAC_PROC_ID to movableDBoxProc. (This enumerated type is defined in the Macintosh include file **Windows.h**.)
2. Create your dialog as a WD_MODAL type with xvt_dlg_create_*.
3. In order to avoid the creation of other windows or dialogs as movable modal dialogs, you must reset the attribute to -1. Do this in the E_CREATE for the dialog.

Example: This example shows how to create a movable Macintosh dialog:

```
long xvtmi_dm_post_fontsize(long defaultsize)
{
        ...
        xvt_vobj_set_attr(NULL_WIN, ATTR_MAC_PROC_ID,
                        movableDBoxProc);
        xvt_dlg_create_res(WD_MODAL, DB_FONTSIZE, EM_ALL,
                        xvtmi_cb_fontsize, (long)&response);
        ...
}

static long xvtmi_cb_fontsize(WINDOW win, EVENT *erp)
{
        ...
        switch (erp->type) {
        case E_CREATE:
                xvt_vobj_set_attr(NULL_WIN,
                        ATTR_MAC_PROC_ID, -1);
        ...
}
```

2.4.5.3. Automatic Dialog Positioning

The DLOG Rez resources that **xrc** produces are compatible with the System 7 automatic dialog position flags. **xrc** places the following code into the DLOG resource:

```
#if SystemSevenOrLater
        "Dialog Title",
        DIALOG_POSITION
#else
        "Dialog Title"
#endif
```

The constant DIALOG_POSITION is defined if SystemSevenOrLater is defined to 1. By default, XVT/Mac defines SystemSevenOrLater to be 1. If you don't want these options, define SystemSevenOrLater to 0 before including **url.h** in your URL file.

The DIALOG_POSITION constant must be defined to one of the supported Rez dialog position flags:

```
noAutoCenter
alertPositionMainScreen
alertPositionParentWindow
alertPositionParentWindowScreen
```

For backward compatibility, DIALOG_POSITION is initially defined to noAutoCenter for all application dialogs. To make use of the new automatic positioning capability, the URL file must contain code to undefine and redefine DIALOG_POSITION, as shown in the following example.

Example: This example shows how to use the DIALOG_POSITION resource:

```
#if XVTWS == MACWS
#ifdef DIALOG_POSITION
#undef DIALOG_POSITION
#define DIALOG_POSITION alertPositionMainScreen
#endif
#endif

DIALOG DB_RESPONSE, 174, 68, 210, 169
                "String Response Dialog" MODAL INVISIBLE
BUTTON DLG_OK,          137, 136, 60, 20   "OK" DEFAULT
BUTTON DLG_CANCEL,  64, 136, 60, 20   "Cancel"
TEXT 4, 13, 10, 184, 80
EDIT 5, 13, 103, 184, 16

#if XVTWS == MACWS
#ifdef DIALOG_POSITION
#undef DIALOG_POSITION
#define DIALOG_POSITION noAutoCenter
#endif
#endif
```

You need to reset the DIALOG_POSITION constant after the URL DIALOG statement only if you have subsequent URL DIALOG statements for which the automatic dialog position should not be applied.

See Also: For more information on automatic dialog positioning and what positioning flags are available, see the "Dialog Manager" chapter in *Inside Macintosh: Macintosh Toolbox Essentials*.

2.4.5.4. Color Dialogs

Your application can access the dialog color resources to create non-portable colorized dialogs.

Tip: To create non-portable colorized dialogs in resources:

Create a dctb resource for each DLOG resource of the dialogs for which you want to provide a color table.

The dctb resource provides a color description of the dialog's window attributes.

Tip: To add color to the dialog controls:

Create an ictb resource for the DITM resource of the dialog for which you want to add color controls.

You can add this information in a #transparent statement in URL containing Rez descriptions, or you can add the resource directly to the binary resource using **ResEdit**. There is no predefined type for the ictb resource in **ResEdit**, so the data has to be coded in hex format.

Note: XVT/Mac does not currently support the font and style tables that can be part of an ictb resource.

Example: This example shows a dialog color table resource in Rez format. The example creates a dctb resource and ictb resource for the About dialog in a fictitious program. The code creates the resources in an URL #transparent statement.

The About dialog has five controls. The ictb resource must have an entry matching the DITL resource for each control. However, you do not have to set colors for each item.

The ictb resource starts with color table length and color table offset pairs for each item. If the color table length or offset is 0, the default control color table is used for that item. Following the item pairs are the actual color tables.

```
#if XVTWS == MACWS
#transparent $$$

resource 'dctb' (DB_ABOUT, "About Dialog Color") {
        {
                wContentColor,
                0xFFFF, 0x0000, 0xFFFF,
                wFrameColor,
                0xFFFF, 0xFFFF, 0xFFFF,
                wTextColor,
                0xCCCC, 0xFFFF, 0xFFFF,
                wHiliteColor,
                0xCCCC, 0x6666, 0xCCCC,
                wTitleBarColor,
                0xCCCC, 0x6666, 0xCCCC
        }
};

data 'ictb' (DB_ABOUT, "About Item Color") {
/*offset            ctb length, ctb offset */
/*0000*/            $"0040 0014"                    /* Item 1 */
/*0004*/            $"0040 0014"                    /* Item 2 */
/*0008*/            $"0040 0034"                    /* Item 3 */
/*000C*/            $"0040 0034"                    /* Item 4 */
/*0010*/            $"0040 0034"                    /* Item 5 */
/*0014*/            $"0000 0000 0000 0002"          /* Color table 1 */
/*001C*/            $"0000 FFFF 0000 FFFF"          /*   Frame color */
/*0024*/            $"0001 AA00 00FF AA00"          /*   Body color */
/*002C*/            $"0002 0000 FFFF FFFF"          /*   Text color */
/*0034*/            $"0000 0000 0000 0002"          /* Color table 2 */
/*003C*/            $"0000 FFFF 0000 0000"          /*   Frame color */
/*0044*/            $"0001 0000 FFFF 0000"          /*   Body color */
/*004C*/            $"0002 0000 FFFF FFFF"          /*   Text color */
};

$$$
#endif
```

See Also: For more information about color dialog resources, see the "Dialog Manager" chapter in *Inside Macintosh: Macintosh Toolbox Essentials*.

Setting Dialog or Window Color at Runtime

You can use the escape function XVT_ESC_MAC_SET_WINDOW_COLOR to modify the color of a dialog or window at runtime. This function lets you set the standard color table for windows and dialogs.

When using the XVT_ESC_MAC_SET_WINDOW_COLOR escape function, create the dialog or window as invisible and call xvt_vobj_set_visible after calling the escape function to set the color. This prevents the window from displaying in the default colors and then changing them when the application calls the escape function.

The WINDOW argument can specify any XVT window or dialog, but not a control. To set colors on controls, use the portable functions xvt_ctl_set_colors and xvt_win_set_ctl_colors.

The escape function XVT_ESC_MAC_SET_WINDOW_COLOR takes pointers to Macintosh RGBColor structures. Any of these pointers can be NULL, in which case the corresponding default window color value is used. You must include the Macintosh header file **QuickDraw.h** to define the RGBColor structure.

Example: Here is an example of how to set a window's color at runtime:

```
#if (XVTWS == MACWS)
#include "QuickDraw.h"
{
RGBColor contentColor, frameColor, textColor,
        hiliteColor, titleBarColor;

contentColor.red = 0x0000;
contentColor.green = 0x0000;
contentColor.blue = 0xFFFF;
frameColor.red = 0xFFFF;
frameColor.green = 0x0000;
frameColor.blue = 0x0000;
textColor.red = 0x0000;
textColor.green = 0xFFFF;
textColor.blue = 0x0000;
hiliteColor.red = 0xFFFF;
hiliteColor.green = 0xFFFF;
hiliteColor.blue = 0x0000;
titleBarColor.red = 0xFFFF;
titleBarColor.green = 0x0000;
titleBarColor.blue = 0xFFFF;
xvt_escape(XVT_ESC_MAC_SET_WINDOW_COLOR, win,
        &contentColor,  &frameColor, &textColor,
        &hiliteColor, &titleBarColor);
}
#endif
```

See Also: See the description for XVT_ESC_MAC_SET_WINDOW_COLOR in Appendix A.

2.4.6. Control Resources

The following subsections describe rules you must follow when creating controls for use with XVT/Mac.

2.4.6.1. Control "Ground" Rules

XVT/Mac has two important rules concerning the numbering of controls:

- The default button must be the first control listed, and its ID must be 1. It is often labeled **OK**, but it doesn't have to be.
- The **Cancel** button (if you have one) must be next, so its ID is 2.

Note: For your About dialog, you can name *either* the **OK** button or the **Cancel** button **Help**, in order to access the help system.

Since the IDs for the first two buttons will be 1 and 2, respectively, you can use XVT's predefined symbols DLG_OK and DLG_CANCEL in your event handler.

Tip: The rest of the items (static text, scrollbars, etc.) can follow in any order. Once you choose an order, use symbolic constants for the controls and don't change the order. If you use the same symbols in dialog resources on other platforms, the same event handler function can support multiple versions of the dialog.

All items are defined as type Control, rather than the actual control type. The actual control type is stored in the CNTL resource specified by the item. This gives XVT more flexibility in designing and creating the controls. If XVT/Mac finds another item type in the dialog item list, it tries to convert it to the appropriate control at load time. All item types are supported, except userItem types.

XVT/Mac does not use the Macintosh Toolbox Dialog Manager. Instead, all dialogs are created as windows with controls when the resource is loaded.

In the file **xvt_mctl.h**, XVT/Mac defines constants for each control definition procedure resource ID, such as xvtStaticTextProc, and control flags, such as MAC_FLAG_NATIVE_JUST. Some of the keywords, such as dBoxProc, visible, and NoGoAway are defined by Rez in the file **Types.r**.

2.4.6.2. Rez Definition of the CNTL Resource

XVT/Mac stores information in certain fields in the control resource record. To get special features in the XVT controls, you may need to set these fields.

The Rez definition of the CNTL resource is:

```
type 'CNTL' {
        rect;                           /* boundsRect */
        integer;                        /* value */
        byte invisible, visible;        /* visible */
        fill byte;
        integer;                        /* max */
        integer;                        /* min */
        integer                         /* procID */
        longint;                        /* refCon */
        pstring;                        /* title */
};
```

The following sections provide details about some of the CNTL resource fields.

CNTL Resource Definition Fields: procID

Tip: To create an XVT control, set the procID field of the CNTL resource to one of the following:

pushButProc	Push button control
radioButProc	Radio button control
checkBoxProc	Check box control
scrollBarProc	Scrollbar control
xvtStaticTextProc	Static text control
xvtIconProc	Icon control
xvtPictProc	Picture control
xvtListBoxProc	List box control
xvtTextEditProc	Edit control
xvtListButtonProc	List button control
xvtListEditProc	List edit control
xvtGroupBoxProc	Group box control
xvtNotebookProc	Notebook control

CNTL Resource Definition Fields: value

For controls, the value field of the CNTL resource passes XVT control flags to all XVT controls:

MAC_FLAG_DISABLED
: Initially disables the control. It is available for all controls.

MAC_FLAG_CHECKED
: Initially checks the radio button or check box controls.

MAC_FLAG_LEFT_JUST
MAC_FLAG_CENTER_JUST
MAC_FLAG_RIGHT_JUST
MAC_FLAG_NATIVE_JUST
: Set the text or title justification for any control except scroll, icon, or picture controls. Some controls, such as push buttons, ignore these flags since the native control does not support this feature.

MAC_FLAG_READONLY
: Sets the selection mode for a list box control to “no selections allowed”. (The default setting is single selection mode.)

MAC_FLAG_MULTIPLE
: Sets the selection mode for a list box control to “multiple selections allowed”. (The default setting is single selection mode.)

MAC_FLAG_MULTILINE
: Sets an edit control to multiple lines for the Macintosh only. Allows Return characters to be entered in the edit control field. For compatibility with other platforms, this control flag overrides the default, which is a single line edit control.

MAC_FLAG_WORDWRAP
: Automatically creates a new line with word wrap when the edit control field is filled. Using MAC_FLAG_WORDWRAP implies MAC_FLAG_MULTILINE. You do not need to set both.

CNTL Resource Definition Fields: refCon

The refCon field of the CTRL resource passes the application control ID for all XVT controls. The value must be greater than zero.

CNTL Resource Definition Fields: min and max

Different controls use the min and max fields in various ways. For static text, list box, edit, and group box controls, the max field is a Macintosh font number and the min field is the font size. For an icon or picture control, the max field is the ICON or PICT resource, respectively, to load for the control. For a scrollbar control, the min and max fields are the initial range values. Other controls ignore these fields.

CNTL Resource Definition Fields: title

The title field of the CTRL resource is normally used for control titles or is ignored. For edit and list edit controls the title is the control's initial text for the edit field.

2.4.7. Menu Resources

The following subsections describe rules you must follow when creating menus for use with XVT/Mac.

2.4.7.1. Creating Macintosh-specific Menus

You might want different menus on the Macintosh version of your application than you have on other platforms. This does not limit portability as long as the items that appear in only one version have unique tags. Define all tags in header files and make sure the switch statement of menu tags in your E_COMMAND event code has a case for every tag.

Tip: You can define an item on a Macintosh menu that you don't use on another platform (such as **Page Setup** on the **File** menu). Define the item normally in your Macintosh URL script. To omit the item on other platforms, use conditional compilation in URL. At the URL level, consecutive numbering of menu tags is unimportant.

2.4.7.2. Menu Item Numbering

It's best to use XVT-Design to create menus in URL, but if you need to create menus in Rez or with resource editing tools, you must follow specific rules. To maintain the user-tag-ID to Macintosh-ID mapping, **xrc** creates an XVTM type resource for each MENU resource. If this resource is not found, the menu items are numbered consecutively starting from the menu ID. For example, the second item on a menu with ID 600 would be 602.

However, if a menu item refers to a hierarchical menu, then the menu item is tagged with the submenu's ID. For example, if the third item of menu 600 refers to menu 700, the item is tagged 700. If the fourth item is not hierarchical, its tag is 604, effectively skipping 603.

Note: If you use a utility such as **ResEdit** to modify menus interactively, be sure you number the items and controls correctly and list them in the correct order. If in doubt, use **ResEdit** to examine the resource fork you've created.

2.4.7.3. "Quit" Menu Item

For a Mac OS X compliant look and feel, XVT will automatically relocate the "Quit" menu item located under the "File" menu to the Application menu.

2.4.7.4. Color Menus

To support color menus, XVT/Mac attempts to load the mctb menu color table resource for each menu. This resource is not supported in URL and must be created in Rez or with a resource editing tool.

The application can access the menu color resources to create non-portable colorized menus. To do this in resources, create an mctb resource for each MENU resource of the menus for which you want to provide a color description. You can add this information in a #transparent statement in URL with Rez descriptions, or you can add the resource directly to the binary resource using **ResEdit**.

Example: This example (an mctb resource defined in a URL file #transparent statement) sets the color table for the **File** menu and a color table for the **New** item on the **File** menu.

```
#if XVTWS == MACWS
#transparent $$$
resource 'mctb' (M_FILE, "File Menu") {
        {
        M_FILE,                         /* Set the File menu */
        0,                              /* default for menu */
        {
                0xFFFF, 0x0000, 0xFFFF,         /* menu title RGB color */
                0xFFFF, 0xFFFF, 0xFFFF,         /* menu bar RGB color */
                0xCCCC, 0xFFFF, 0xFFFF,         /* menu item default RGB
                                                   color */
                0xCCCC, 0x6666, 0xCCCC          /* menu background RGB
                                                   color */
        },
        M_FILE,                         /* Set the File menu */
        1,                              /* Menu Item 1 - "New" */
        {
                0xFFFF, 0x0000, 0x0000,         /* menu item mark RGB
                                                   color */
                0x0000, 0xFFFF, 0x0000,         /* menu item text RGB
                                                   color */
                0x0000, 0xFFFF, 0xFFFF,         /* menu item accelerator
                                                   RGB color */
                0x0000, 0x0000, 0xFFFF          /* menu item background
                                                   RGB color */
        }
        }
};
$$$
#endif
```

Note: You can also set or change menu colors at runtime using the Macintosh Toolbox functions GetMCInfo and SetMCEntries.

See Also: For more detailed information on color menu resources, see the "Menu Manager" chapter in *Inside Macintosh: Macintosh Toolbox Essentials*.

2.4.7.5. Menu Accelerators

Macintosh menu accelerators or keyboard equivalents must always begin with the Command key. Any printable ASCII character can be used as a menu accelerator without limiting portability.

You can use some modifier keys but not others:

Shift key
Invalid (Macintosh doesn't differentiate between uppercase and lowercase).

Control key
Invalid.

Command key
Valid. Specify this key using the alt keyword.

Option key
Valid. Use the non-portable character associated with an Option key sequence. Type this character directly into the URL accel statement. **xrc** compiles this character into the native Macintosh resource.

Example: The following URL statement sets the menu accelerator for the menu item M_BAKE_APPLE_PIE to Option-P:

```
#if XVTWS == MACWS
    accel M_BAKE_APPLE_PIE "¼" alt
#endif
```

See Also: For more information, see the E_CHAR event and the accel statement in the online *XVT Portability Toolkit Reference*.

2.4.7.6. Balloon Help Menu Access

Note: Support for Balloon Help has been discontinued by Apple for Mac OS X and greater. This section is included if Balloon Help is to be implemented in a "Classic" Carbon application

You can automatically add menu items to your application's Balloon Help menu by creating a menu in URL with the ID M_HELP. On other platforms, you would add the M_HELP menu to the menubar as a regular menu. On the Macintosh platform, however, you append the items in the M_HELP menu to the Macintosh Balloon Help menu.

XVT/Mac provides a default M_HELP menu for use with online help. To use the default M_HELP menu, place the item DEFAULT_HELP_MENU in your application's menubar resource(s). If you do not want to use the default M_HELP menu, you must define the NO_STD_HELP_MENU symbol before including **url.h** in your URL file.

Put the M_HELP menu at the menubar level and do not attempt to make it a submenu of another menu. Because the Macintosh operating system does not support submenus on the Balloon Help menu, you should also avoid putting submenus on the M_HELP menu in URL.

Note: Because the help menu must have the ID M_HELP, you can have only one help menu in the resource for all menubars. If you want different menubars to have different items appended to the Balloon Help menu, your application must do this through xvt_menu_get_tree and xvt_menu_set_tree.

When xvt_menu_get_tree is called, XVT/Mac returns the items on the Balloon Help menu as a MENU_ITEM array describing the M_HELP menu at the end of the menubar MENU_ITEM array. If you don't place the M_HELP menu at the end of the menubar in URL, you can't depend on it having the same location when your application calls xvt_menu_get_fetch. This is because the M_HELP menu is always added at the end of the menubar MENU_ITEM array.

As you might expect, xvt_menu_set_tree makes any necessary modifications to the Balloon Help menu items based on changes to the M_HELP menu. If the user selects one of the appended items, an E_COMMAND event is sent to the event handler for the window that owns the current menubar. The E_COMMAND event ID that is sent is the one assigned to the item in URL or through xvt_menu_set_tree.

Note: Don't forget to create the Macintosh native hmnu resources for the items you add to the Balloon Help menu.

Example: This example (based on another example from *Inside Macintosh: Macintosh Toolbox Essentials)* shows how to code help menu resources in URL:

```
MENU M_HELP "Help"
        ITEM M_HELP_SURFWRITER "SurfWriter Help"
        ITEM M_HELP_WIPEOUT "WipeOut Help"

MENU TASK_MENUBAR
        DEFAULT_FILE_MENU
        DEFAULT_EDIT_MENU
        SUBMENU M_HELP "Help"

MENUBAR TASK_MENUBAR

#if XVTWS == MACWS
#transparent $$$ literal
#include "BalloonTypes.r"
resource 'hmnu' (kHMHelpMenuID, "Help", purgeable) {
HelpMgrVersion, 0, 0, 0,
HMSkipItem {
},
{
        HMStringResItem {
        146, 1,
        146, 2,
        146, 3,
        0,0
        },
        HMStringResItem {
        146, 4,
        146, 5,
        146, 6,
        0,0
        },
}
};

resource 'STR#' (146, "My help menu items' strings") {
{
"Displays help for SurfWriter word processor.";
"Displays help for SurfWriter word processor.";
"Not available until you open a SurfWriter document.";
"Closes help for SurfWriter word processor.";
"Displays help for WipeOut typing corrector.";
"Displays help for WipeOut typing corrector.";
"Not available until you open a SurfWriter document.";
"Closes help for WipeOut typing corrector.";
}
};
$$$
#endif
```

2.4.8. Cursor Resources

You can use cursors of your own design, as long as their IDs are greater than 10. Lower numbers are reserved for XVT's standard cursors. You can use your cursors' IDs directly in calls to the xvt_win_set_cursor function. You can load cursors directly into the resource fork with a resource editing tool such as **ResEdit**, or place them in a #transparent statement in URL. You cannot code them in URL.

Example: Here is an example of a cursor resource coded in Rez format:

```
#if XVTWS == MACWS
#transparent $$$
resource 'CURS' (128) {
        $"0000 0F00 0880 1080 1900 2700 2200 4200"
        $"4400 8400 8800 C800 F000 E000 C000 8000",
        $"0000 0000 0000 0000 0000 0000 0000 0000"
        $"0000 0000 0000 0000 0000 0000 0000 0000",
        {15, 0}             /*Hot point*/
};
$$$
#endif
```

The XVT function xvt_win_set_cursor also loads color cursors from crsr resources. Each crsr resource must be a color representation of any CURS resource with the same ID, since XVT/Mac always tries to load the crsr resource first.

If the crsr resource isn't found and ATTR_MAC_HAVE_COLOR_QUICKDRAW is TRUE, XVT/Mac tries to load a CURS resource with the specified ID.

2.4.9. Control Icon Resources

You can place icons as controls in dialogs and windows. When you create a dialog or window, the icon is created just as any other control would be.

Tip: To create icons in dialog or window resources:

1. Create your icon using a resource editing tool such as **ResEdit**.
2. Add an ICON statement in your URL file; a sample ICON statement is shown below:

   ```
   ICON ICON_RID 200, 300, 75, 50 123
   ```

   where the statement consists of:

   - The keyword ICON
   - The resource ID

- The x, y, width, and height specifications
- The native icon resource ID

3. Do one of the following:

   To your **.url** file, add a #transparent statement that loads the icon from a binary resource file with the Rez include statement. For example, to include the binary resource file **icon.rsrc**, containing one or more icon resources:

   ```
   #if XVTWS == MACWS
   #transparent $$$
   include "icon.rsrc";
   $$$
   #endif
   ```

   -OR-

   DeRez your icon and place the resulting Rez text for the icon in a #transparent statement in your **.url** file. (To learn how to decompile resources with **DeRez**, see your *MPW Command Reference* or other compiler reference.)

You can also create icons in windows and dialogs using the function xvt_ctl_create_def. (The function xvt_ctl_create does not support the creation of icons.

Tip: To create an icon control using xvt_ctl_create_def:

1. Create the icon using a resource editing tool such as **ResEdit**.
2. Add a #transparent statement to your URL file.
3. Create a WIN_DEF structure with the type specified as WC_ICON.
4. Specify the icon_id in the v.ctl.icon_id portion of the WIN_DEF structure. This icon_id must be the same as the ICON or cicn resource ID. Each cicn resource must be a color representation of any ICON resource with the same ID, since XVT/Mac always tries to load the cicn resource first. For the URL ICON statement described at the beginning of this section, setting the icon_id in the WIN_DEF structure looks like this:

   ```
   windef[0].v.ctl.icon_id = 123;
   ```

2.4.10. Drawn Icons Resources

You cannot directly define an icon in URL, but you can put a Rez statement in a #transparent statement to define one. You can also create the icon using a resource editing tool such as **ResEdit** and include the binary resource in a #transparent statement using the Rez include statement.

Example: This example shows how to code an icon resource in Rez format:

```
#if XVTWS == MACWS
#transparent $$$
resource 'ICON' (130) { /* Icon resource with ID 130 */
        $"F056 8E0F 0FD7 B078 0075 6F84 0FFF F1F2"
        $"07FC 7E00 F078 3FF8 0F98 3FC6 0079 20A0"
        $"03D8 3CA0 0C3C 2250 001E 4A50 0021 E220"
        $"1F40 9E00 00C8 6100 00A0 2098 07D4 22E0"
        $"1880 2080 20B4 24F0 C0C0 408C 01A0 E100"
        $"011F DE00 0209 C000 0409 C76A 0809 C54C"
        $"3019 E74A 4019 556A 0021 2800 0030 9400"
        $"0031 CA00 0070 AA00 0060 A200 0080 9C00"
};
$$$
#endif
```

You can use your icon IDs when you call xvt_dwin_draw_icon. xvt_dwin_draw_icon also loads color icons from cicn resources. Each cicn resource must be a color representation of any ICON resource with the same ID, since XVT/Mac always tries to load the cicn resource first.

If ATTR_MAC_HAVE_COLOR_QUICKDRAW is TRUE and the cicn resource isn't found, XVT/Mac attempts to load an ICON resource with the specified ID.

2.4.11. Finder Icon Resources

A complete set of finder icons includes a black and white icon, an icon mask, and 8-bit color icons and 4-bit color icons for both the 16-by-16 and 32-by-32 sizes. To provide a complete set, you need to specify six different resources:

- ICN# for large black and white icon and mask
- ics# for small black and white icon and mask
- icl8 for large 8-bit color icons
- ics8 for small 8-bit color icons
- icl4 for large 4-bit color icons
- ics4 for small 4-bit color icons

The ICN# and ics# resources also contain an icon mask that is used with the icon family to indicate user selection.

The easiest way to create an icon family is to use **ResEdit**. Binary icon resources may be included from a **ResEdit**-generated file by using a Rez include statement inside a URL #transparent. Alternatively, you may use **DeRez** to convert the binary icon to a text form that can be inserted into a URL #transparent.

Example: This example shows how to code an icon list resource in Rez format:

```
#if XVTWS == MACWS
#transparent $$$
resource 'ICN#' (140) { /* Icon list resource - ID 140 */
        { /* The first icon */
        $"F056 8E0F 0FD7 B078 0075 6F84 0FFF F1F2"
        $"07FC 7E00 F078 3FF8 0F98 3FC6 0079 20A0"
        $"03D8 3CA0 0C3C 2250 001E 4A50 0021 E220"
        $"1F40 9E00 00C8 6100 00A0 2098 07D4 22E0"
        $"1880 2080 20B4 24F0 C0C0 408C 01A0 E100"
        $"011F DE00 0209 C000 0409 C76A 0809 C54C"
        $"3019 E74A 4019 556A 0021 2800 0030 9400"
        $"0031 CA00 0070 AA00 0060 A200 0080 9C00"
        , /* 2nd icon is mask for finder bundle */
        $"1FA9 71F0 F028 4F87 FF8A 907B F000 0E0D"
        $"F803 81FF 0F87 C007 F067 C039 FF86 DF5F"
        $"FC27 C35F F3C3 DDAF FFE1 B5AF FFDE 1DDF"
        $"E0BF 61FF FF37 9EFF FF5F DF67 F82B DD1F"
        $"E77F DF7F DF4B DB0F 3F3F BF73 FE5F 1EFF"
        $"FEE0 21FF FDF6 3FFF FBF6 3895 F7F6 3AB3"
        $"CFE6 18B5 BFE6 AA95 FFDE D7FF FFCF 6BFF"
        $"FFCE 35FF FF8F 55FF FF9F 5DFF FF7F 63FF"
        }
};
$$$
#endif
```

2.4.12. String Resources

You can access STR and STR# resources (portably) with the XVT functions xvt_res_get_str and xvt_res_get_str_list, respectively.

Example: This example shows sample string and string list resources coded in Rez format:

```
resource 'STR ' (210, purgeable) {
        "Test string to read with xvt_res_get_str"
};

resource 'STR#' (260, purgeable) {
        {
                "Alabama",
                "Alaska",
                "Arizona",
                "Arkansas",
                "California",
                "Colorado",
                        ...
                "West Virginia",
                "Wisconsin",
                "Wyoming"
        }
};
```

When calling the function xvt_res_get_str_list, you should specify start and end resource IDs so that your call will work on other platforms, even though the end resource ID is not used on the Macintosh. In this example, the start ID is the same as the ID for the STR# resource (260) and the imaginary end ID is (260 + 50 – 1), or 309. The xvt_res_get_str_list call looks like this:

```
SLIST x;
...
if ((x = xvt_res_get_str_list(260, 309)) == NULL {
        ... handle error ...
}
```

Note: You can almost always code string resources in URL.

2.4.12.1. International Strings

All XVT Portability Toolkit functions have moved internal English strings to the resource file to facilitate localization of applications. Each platform defines a constant for each string's resource ID. The default file **uengasc.h** in the **include** folder contains the XVT/Mac string constants. Other language and character codeset localizations can be found in the same location.

2.4.12.2. String Resource IDs

During xvt_app_create, XVT/Mac locates and loads the following string resource IDs to replace internal XVT strings (if they exist in the resource):

MAC_STR_HELP_ID
: See ATTR_MAC_STR_HELP in Appendix A.

MAC_STR_STYLEM1_ID
: See ATTR_MAC_STR_STYLE_MENU1 in Appendix A.

MAC_STR_STYLEM2_ID
: See ATTR_MAC_STR_STYLE_MENU2 in Appendix A.

MAC_STR_STYLEM3_ID
: See ATTR_MAC_STR_STYLE_MENU3 in Appendix A.

MAC_STR_ABOUT_ID
: Replaces the text “About %s...” in the item on the Apple Menu that brings up the About Box dialog, as in “About XVT...”. The %s format item is replaced with the application name.

MAC_STR_CLICK_ID
: Replaces the text “Click mouse to continue.” that is displayed in an XVT Alert dialog (created by xvt_dm_post_fatal_exit or xvt_dm_post_message).

MAC_STR_SELECT_ID
: Replaces the text "Select \322%s\323" that is displayed by the Open File Dialog when selecting directories. The %s format represents the current directory name selected.

MAC_STR_FIXED_FONT_ID
: Allows the application to define the font name specified by the font define XVT_FFN_FIXED. The internal default font used is “monaco”.

MAC_STR_LOW_MEM_WARNING_ID
: Replaces the text “There is very little memory available. Please increase the preferred memory size in the application’s Get Info window.” displayed when the XVT application is low on available memory.

MAC_STR_NOTIFICATION_ID
: Replaces the text “The application \322%s\323 needs your attention.\n\n” displayed by the Mac Notification Manager when the application needs attention. The %s format item is replaced with the application name.

2.5. XVT's Encapsulated Font Model

2.5.1. Font Terminology

This section uses the following XVT-defined terms to describe XVT's encapsulated font model:

Physical font
A particular *implementation* of a font as installed on the window system on which an application is running.

Logical font
A *description* of a desired physical font, to a degree of specificity ranging from just a typeface family name or size to a complete description that specifies a particular physical font. A logical font has both portable and non-portable attributes. It is identified by an object of type XVT_FNTID.

2.5.2. Native Font Descriptors

To specify a particular physical font, your application can use a native font descriptor, which is a string of data fields. You can include this string as a parameter to xvt_font_set_native_desc, or in URL as part of a FONT or FONT_MAP statement.

The native font descriptor string contains the following fields:

- The native window system and version of the XVT encapsulated font model (the current version is "01").
- Platform-specific fields that the XVT Portability Toolkit decodes and uses to uniquely specify a native font. The fields describe specific attributes of a native font. Each field is separated by a slash, "/".

The native font descriptor string, then, has this format:

"<system and version>/<field1>/<field2>/<field3>/
...<fieldn>"

See Also: For more information about specifying fonts, see the "Fonts and Text" chapter in the *XVT Portability Toolkit Guide*.

2.5.2.1. XVT/Mac Font Descriptor Version Identifier

For XVT/Mac, the font descriptor version identifier format is MAC<vers>. In this release of XVT/Mac, the font descriptor version number is "01," so the font descriptor version identifier is MAC01.

2.5.2.2. XVT/Mac Font Fields

On the Macintosh, four attributes specify a font: family, face (style), size, and mode. The mode attribute is already addressed by the DRAW_CTOOLS mode member. Consequently, the native font descriptor string contains just the family, face, and size. For XVT/Mac, the string should look like this:

"MAC01/<family>/<face>/<size>"

For <face>, you can use plain, or any of the following valid values, in any order separated by "-":

bold
italic
underline
outline
shadow
condense *or* extend (but not both)

Example: This string shows a valid XVT/Mac native font descriptor string:

"MAC01/times/bold-italic/*"

Note: An asterisk (*) in a native font descriptor string indicates a wildcard condition in which any value is acceptable for that particular field.

3

DEVELOPMENT ENVIRONMENT

3.1. Introduction

This chapter gives detailed information on building XVT/Mac applications. The Macintosh development environment supports one compiler: Metrowerks CodeWarrior C/C++.

*XVT-Design and XVT-Architect generated IDE project files automatically include the appropriate options for the compiler. For up-to-date information regarding compiler settings and libraries, see the **Readme** file in the **doc** folder.*

Your compiled application consists of one linked application file containing two forks:

Data Fork
Contains executable image for PowerPC-based applications.

Resource Fork
Contains the executable image and other resources, including descriptions of your windows, dialogs, controls, menubars, and strings.

Each development environment requires that resources be compiled for the resource fork. At runtime, you simply run the linked application.

See Also: Resource compilers are discussed in detail in section 3.3.

3.1.1. Include Files

XVT-Design and XVT-Architect generate code that automatically includes all necessary header files.

To build XVT applications, you must include the XVT-specific header file, **xvt.h** in addition to any other application-specific header files.

xvt.h automatically includes several standard header files. You do not have to include the following header files explicitly in your application, although doing so is harmless:

ctype.h	**stdio.h**
fcntl.h	**stdlib.h**
stdarg.h	**string.h**
stddef.h	**time.h**
Types.h	**unix.h**

Most of these header files are ANSI C files and are fully portable. However, the code you write using information in these files may be non-portable because of variations in the ANSI C implementations. If so, you need to conditionally compile around that code.

Do not assume Macintosh header files have been included. You should explicitly include the Macintosh header files you need before you include **xvt.h**.

The Metrowerks CodeWarrior C/C++ compiler requires the following libraries and files:

- The XVT API library
- The Metroworks Standard Libraries (MSL)
- The Mac Toolbox library
- Your application object files

XVT also requires either the XVT text edit libraries or the XVT text edit dummy library. Additionally, if you want to use hypertext online help with your application, you must use either the bound XVT help viewer library or the standalone XVT help viewer library.

See Also: For up-to-date information on library names, see the **Readme** file in the **doc** folder. For more information on the required libraries, see section 3.2.2 (Metrowerks CodeWarrior C/C++).

3.2. Metrowerks CodeWarrior C/C++ Development Environment

The following subsections contain detailed information on building XVT/Mac applications in Metrowerks CodeWarrior C/C++ on both PowerPC platforms.

3.2.1. Environment Options: Metrowerks CodeWarrior C/C++

Note: Although this information is correct at publishing time, for the most up-to-date information, see the **Readme** file in the **doc** folder.

When using XVT with the Metrowerks CodeWarrior C/C++ compiler, you must set several project settings, as shown below.

Note: Settings labeled *recommended* are not required, but XVT recommends that you use them.

Tip: To set project settings:

1. Select **Edit=>ProjectName Settings**—then select the **Target:Access Paths** item from the preference dialog. Add the XVT **include** and **lib** folders to the user search path.
2. Select the **Target:PPC Target** or **Target:68K Target** item. Set the SIZE Flags value as indicated in the table below. Set your application name and desired heap sizes as well.

Group:	Option:	Value:
'SIZE Flags'	acceptSuspend ResumeEvents	On (*required*)
'SIZE Flags'	canBackground	On (*recommended*)
'SIZE Flags'	doesActivateOn FGSwitch	On (*recommended*)
'SIZE Flags'	is32BitCompatible	On (*recommended*)
'SIZE Flags'	isHighLevelEvent Aware	On (*required*)
'SIZE Flags'	localAndRemoteHL Events	On (*required*)

3. Select the **Language Settings: C/C++ Language** item. Set the Language Info values as indicated in the table below.

Group:	Option:	Value:
Language Info	ARM conformance	On (*required*)
Language Info	Enable C++ Exceptions	On (*required*)
Language Info	Require Function Prototypes	On (*recommended*)
Language Info	Enable bool Support	On (*required*)
Language Info	Enable wchar_t Support	On (*required*)
Language Info	Enums Always Int	Off (*required*)

4. Select the **Code Generation:PPC Processor Processor** item. Set the Code Model value, Struct Alignment value, and Info values as indicated in the table below.

Group:	Option:	Value:
PPC Processor	Struct Alignment	68K (*required*)

5. Select the **Linker:PPC Linker** item. Set the Link Options/Info as indicated in the table below.

Group:	Option:	Value:
Link Options	Dead-strip Static Initialization Code	Off (*required*)

Note: If an option is not noted as required in the above tables, you may set it as desired for your application.

Compiler Optimization Flag

XVT provides a compiler optimization flag, XVT_OPT, for runtime optimization of the PTK. This flag is described further in the *XVT Portability Toolkit Guide*. To use the flag with the Metrowerks CodeWarrior C/C++ compiler, you must setup and use the prefix file for the compiler:

1. To create a prefix file, edit a new file. Enter the following text in the file to define XVT_OPT:

```
#define XVT_OPT 1
```

2. Save the file and name it as you like, “XVT Prefix” for example.
3. Select **Edit=>ProjectName Settings...**—then select the **Language Settings:C/C++ Language** item from the preference dialog. Enter your prefix file in the Prefix File edit field.

Now recompile and link your application. To remove the optimization, simply remove the prefix file name from the Prefix File edit field described above.

3.2.2. Link Libraries

When using the Metrowerks CodeWarrior C/C++ compiler, the XVT library is distributed as the following library files:

XVTmPPCmwAPI.lib	Core library (*required*)
XVTmPPCmwTE.lib	Text edit library; *required* when you use the bound help viewer or any XVT functions starting with "xvt_tx_"
XVTmPPCmwTES.lib	Dummy text edit library; use instead of **XVTmPPCmwTE.lib**
XVTmPPCmwHI.lib	The online help IPC viewer application interface library
XVTmPPCmwHB.lib	The online help viewer application interface bound to the application
XVTmPPCmwPWR.lib	XVT-Power++ class library
XVTmPPCmwRW.lib	RogueWave Tools.h++ class library

All XVT libraries are located in the **lib** folder.

Tip: Add the XVT libraries to your project as follows:

1. Add **XVTm*mwAPI.lib** to your project using the **Add Files...** command on the **Project** menu.
2. If you are using the text edit object, add the text edit library **XVTm*mwTE.lib**; otherwise, add the dummy text edit library, **XVTm*mwTES.lib**, found in the **lib** folder.
3. If you are using online help, add one of the libraries, depending on whether you are using the bound help viewer or the standalone IPC help viewer:

   The bound help viewer requires the **XVTm*mwHB.lib** library.

   The standalone IPC help viewer application requires the **XVTm*mwHI.lib** library.
4. In addition to the XVT libraries, add the following standard libraries that come with Metrowerks CodeWarrior C/C++:

   PowerPC-based Macintoshes C and C++ applications:

InterfaceLib, **MathLib**, **MSL C.PPC.Lib**, **MSL RuntimePPC.Lib**, **MSL SIOUX.PPC.Lib**

PowerPC-based Macintoshes C++ applications:

MSL C++.PPC.Lib

3.2.3. For Source Customers Only: XVT/Mac Development Environment

This section contains information pertinent to XVT/Mac source customers. If you have purchased the XVT/Mac binary product, you can skip this section.

3.2.3.1. Building the XVT/Mac Libraries

XVT supplies the following Metrowerks CodeWarrior C/C++ project files for the XVT/Mac Portability Toolkit libraries:

XVT*API.mu for library **XVTm*mwAPI.lib**
XVT*TE.mu for library **XVTm*mwTE.lib**
XVT*TES.mu for library **XVTm*mwTES.lib**
XVT*HB.mu for library **XVTm*mwHB.lib**
XVT*HI.mu for library **XVTm*mwHI.lib**

Tip: To build the XVT/Mac libraries:

1. Create **lib** folder if it doesn't exist.
2. Select **Project=>Reset Project Entry Paths**.
3. Select **Project=>Research for Files**
4. Select **Project=>Make**. Do this for each Metrowerks CodeWarrior C/C++ project.

The makefile places the finished libraries in the **lib** folder.

3.2.3.2. Building Utility Programs

For source customers, XVT/Mac supplies projects for the utility programs **xrc**, **errscan**, **helpc**, and **helpview**. For each of these utility programs, first build the resources, and then build the application executable.

Tip: For example, to build resources for **xrc**:

1. Use the **xrc** supplied with the XVT Portability Toolkit to compile the file **xrcxvt.url** in the **src:xrc** folder into **xrcxvt.r**, the native resource file.

Tip: To build **xrcxrc**:

1. Open the **xrcppc.mu** project in the **src:xrc** folder.
2. Select **Project=>Reset Project Entry Paths**.
3. Select **Project=>Research for Files**.
4. Select **Project=>Make**.

Repeat steps 1 – 4 for each utility you need to build. The makefile puts the finished utility programs in the **bin** folder.

Note: Since these utility programs are XVT applications, make sure that the XVT/Mac libraries are built and installed in the expected location prior to building these programs.

See Also: Refer to the **Readme** file shipped with the XVT/Mac Portability Toolkit for the location of utility program source files.

3.3. Compiling Resources

3.3.1. Using the xrc Interactive Interface

On the Macintosh platform, **xrc** is an interactive application. On other platforms, xrc is used as the line compiler.

Tip: To start **xrc**:

1. Check that your **xrc.opt** file exists in the **bin** folder and contains the options needed to compile your URL file.

   xrc.opt tells xrc.app where to look for header files. Replace the sample path with the full path of your XVT include folder and enclose the pathname in single or double quotes, as follows:

   ```
   -i 'HD:xvtdsc56:mac_ppc:include:'
   -d HIGH_LEVEL_EVENT_AWARE
   ```

   If you have other include folders you want **xrc** to search, add another -i option on a new line of **xrc.opt** for each include folder. Begin the new line with -i, followed by a space, then the include folder pathname enclosed in single or double quotes. Repeat this process for each include folder you want **xrc** to search.

2. Double click on the **xrc** icon.
3. Select **Translate=>Rez**. A standard file dialog appears.
4. From the file dialog, select the URL file to be translated. A second standard file dialog appears.
5. Enter the output file name for the Rez script and select **Save**. The translation begins.

By default, **xrc** appends **.r** to the URL file root name.

During the translation, the status window displays the current file and line number. If **xrc** detects any errors in the file it is compiling, it displays them in pop-up text windows, but only the first five errors are reported. You can edit the input file while viewing the error window. Then either close the error window manually, or let **xrc** close it when you quit or begin another translation.

3.3.1.1. Using Drag and Drop with xrc

The output is always in Rez format. The output filename is the URL file root name with **.r** appended.

Caution: If an output file already exists, **xrc** overwrites it. Once **xrc** has started processing, you cannot cancel the operation until all files have been processed or an error occurs.

3.3.2. Macintosh Resource Compilers

Before reading this section, familiarize yourself with the documentation for the compiler supplied with your development environment (the **Rez** compiler for Metrowerks CodeWarrior C/C++).

3.3.2.1. Metrowerks CodeWarrior Rez Compiler

The CodeWarrior **Rez** compiler is part of the CodeWarrior build process.

Tip: To compile your Rez script files:

1. In the **CodeWarrior IDE** application, open your application project file.
2. Select **Project=>Add Files**.
3. Select your **.r** file and click the **Add** button followed by the **Done** button.
4. From the **Project** menu, select the **Bring Up To Date** or **Make** item to compile the **.r** file as well as all the other files.

There are currently some limitations when using the CodeWarrior **Rez** compiler. There is no method available to set Rez compiler options through **Metrowerks CodeWarrior**. Thus it is not possible to compile resources translated for Japanese or other multibyte character codesets.

See Also: For more information about **Metrowerks CodeWarrior Rez**, see your *Metrowerks CodeWarrior C/C++ User's Guide*.

3.4. Building Your Application with the Help System

XVT's hypertext online help system requires a help viewer. For XVT/Mac, you can bind the portable viewer to the application or run it as a separate (standalone) executable.

See Also: For information on the help viewers, see the "Hypertext Online Help" chapter in the *XVT Portability Toolkit Guide*.
For information on the portable help compiler command options, refer to the online *XVT Portability Toolkit Reference* .

3.4.1. Portable Viewers

XVT/Mac provides the XVT portable hypertext help viewer in both bound and standalone forms. For both forms, you must use the XVT help compiler **helpc** to produce XVT-portable binary help files for the help viewer to use.

3.4.2. Using the Helpc Interactive Interface

On the Macintosh platform, Helpc is an interactive application. On other platforms, it is used as a line compiler.

Tip: To start **Helpc**:

1. Double click on the **Helpc** icon.
2. Select **Compile=>Set Options** to invoke the help compiler options dialog.
3. From the help compiler options dialog, select the **Find** button in the **Source File Selection** group box. From the file selection dialog that appears, select the help source file to be compiled.
4. From the help compiler options dialog, select the **Find** button in the **Include Directories** group box. From the file selection dialog that appears, select **include**.

   Repeat this step for any other required help include directories.
5. From the help compiler options dialog, select any other desired options.
6. From the help compiler options dialog, select the **Apply** button.
7. Select **Compile=>Begin Compile** to compile the help source.

By default, **Helpc** appends **.csc** to the help source file root name.

During the compilation, the status window displays the current status. If **Helpc** detects any errors in the file it is compiling, it displays them in the status window.

Once you have selected your help compiler options and have successfully compiled your help source file, you can save the set of selected compiler options for later use. The saved compiler options are specific to the help source file.

Tip: To save the help compiler options:

From the help compiler **File** menu, select **Save As**. From the file selection dialog that appears, enter a filename for the help compile options to be saved.

Tip: The next time you compile the help file, follow these steps:

1. Double click on the **Helpc** icon.
2. From the **File** menu, select **Open**. From the file selection dialog that appears, select the previously saved help options file. The help compiler status window shows the help source filename for the selected options file.
3. Select **Compile=>Begin Compile** to compile the help source.

3.4.2.1. Bound Viewer

Tip: To bind the help viewer to your application:

Link with one of the following sets of libraries (in addition to the base XVT libraries):

Metrowerks CodeWarrior C/C++:

XVTmPPCmwHB.lib	The online help viewer application interface bound to the application

---

If you are providing context-sensitive help from *modal* XVT windows or dialogs, XVT strongly recommends that you use the standalone help viewer. The bound help viewer is a *modeless* window in XVT. Opening a modeless window from a modal object may result in undefined behavior.

3.4.2.2. Standalone Viewer

Helpview is an application that communicates with XVT applications to provide hypertext help.

Tip: To use **Helpview** with your application:

Link your application with the following library (in addition to the base XVT libraries):

Metrowerks CodeWarrior C/C++:

XVTmPPCmwHI.lib	The online help IPC viewer application interface library

A

APPENDIX A: NON-PORTABLE ATTRIBUTES AND ESCAPE CODES

A.1. Non-portable Attributes

The xvt_vobj_set_attr and xvt_vobj_get_attr functions allow you to manipulate XVT attributes. Non-portable attributes let you fine-tune your application to make it more closely adhere to the look-and-feel of the underlying platform, or to add functionality not provided by the XVT interface. This section provides a list of the non-portable attributes for use with XVT/Mac.

See Also: Additional non-portable attributes may be listed in the **Readme** file in the **doc** folder.

ATTR_MAC_ALWAYS_UPDATE

Description: Normally, an E_UPDATE event is not sent when the update window's update region is empty. However, if this attribute is set to TRUE, E_UPDATE events are sent when update_window is called, even if the update region is empty.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets empty region update event state
xvt_app_create use:	Can use either before or after
Default value:	FALSE
Argument type:	BOOLEAN

ATTR_MAC_BEHIND_WINDOW

Description: Used by the functions xvt_win_create_* to place the window at a different level, rather than always creating a new active window. If you want to create a new WINDOW behind an existing XVT WINDOW, set this attribute to the existing XVT WINDOW before calling xvt_win_create_*. ATTR_MAC_BEHIND_WINDOW has no effect when a modal dialog or modal window is active. Setting this attribute to NULL_WIN creates a new window as the top active window.

Uses win argument:	Yes
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets layer WINDOW for creating new windows
xvt_app_create use:	Must use after
Default value:	NULL_WIN
Argument type:	Value argument is ignored

ATTR_MAC_CHAR_TO_TASK

Description: For portability to other XVT platforms, characters typed while no window has focus are ignored. However, if you want your application to receive these E_CHAR events on the Macintosh only, set this attribute to TRUE, which sends the E_CHAR events to the task window event handler.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sends unused E_CHAR events to TASK_WIN
xvt_app_create use:	Can use either before or after
Default value:	FALSE
Argument type:	BOOLEAN

ATTR_MAC_CONTROL_HANDLE

Description: Given a WINDOW for an XVT control, returns the Macintosh ControlHandle. The values of the control must not be changed directly.

Uses win argument:	Yes
xvt_vobj_get_attr returns:	Macintosh ControlHandle for XVT control WINDOW
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Must use after
Default value:	None
Argument type:	ControlHandle

ATTR_MAC_CTL_DEFER_UPDATE

Description: Setting this attribute to a value of one prior to a series of calls to the function xvt_vobj_set_title or xvt_vobj_move on any control suspends updates to the controls until the attribute is set to zero. The parent window of the control group must be passed using the win argument. Only one parent window at a time is allowed to have this attribute set to one. The application can then set the attribute to zero, causing all the controls to be updated. ATTR_MAC_CTL_DEFER_UPDATE provides a performance enhancement for windows or dialogs with large numbers of controls that must be initialized or set as a group.

Uses win argument:	Yes
xvt_vobj_get_attr returns:	Currently set value for win argument
xvt_vobj_set_attr effect:	Suspends updates to edit controls caused by xvt_vobj_set_title or xvt_vobj_move
xvt_app_create use:	Must use after
Default value:	Zero
Argument type:	long

ATTR_MAC_EVENT_TIME

Description: Specifies the number of ticks of processor usage that your application relinquishes to other applications if no events are pending for it. This is the value passed to the native Macintosh function WaitNextEvent.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets shared processing time
xvt_app_create use:	Can use either before or after
Default value:	One
Argument type:	long

ATTR_MAC_FOREIGN_WIN

Description: Normally, when the user activates a non-XVT window, the current menubar state is saved and the menubar is grayed (disabled). The menubar is restored when the user interacts once again with an XVT WINDOW. However, if you set this attribute to TRUE, XVT is prevented from graying the menubar.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets non-XVT window menubar state
xvt_app_create use:	Can use either before or after
Default value:	FALSE
Argument type:	BOOLEAN

ATTR_MAC_FRONT_WINDOW_FCN

Description: You can define your own C function to implement the functionality equivalent to the Macintosh Toolbox function FrontWindow. Set this attribute to point to your own function to help implement floating windows. Passing a value of NULL sets the function pointer back to the internal function.

Caution: *Never* set the function pointer to point directly to FrontWindow.

Prototype: WindowPtr XVT_CALLCONV1 front_window_function (void)

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Installs front window function, or installs default function if value is NULL
xvt_app_create use:	Can use either before or after
Default value:	Internal default front window function

ATTR_MAC_HAVE_COLOR_QUICKDRAW

Description: Set to TRUE during xvt_app_create if Color QuickDraw (version 2.0 or later) is available.

Uses win argument:	No
xvt_vobj_get_attr returns:	Currently set value
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Must use after
Default value:	FALSE
Argument type:	BOOLEAN

See Also: The attribute ATTR_HAVE_COLOR on page A-26.

ATTR_MAC_HILITE_MODE

Description: Allows applications to draw in the highlight color as selected by the user from the Color Control Panel. When you set this attribute to TRUE before any xvt_dwin_draw_* function is called using M_XOR mode, the object is drawn in the highlight color. For text, the box behind the text, rather then the text itself, is drawn in highlight color. The attribute must explicitly be set to FALSE to discontinue drawing in the highlight color.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Enables/disables highlight color draw mode
xvt_app_create use:	Must use after
Default value:	FALSE
Argument type:	BOOLEAN

ATTR_MAC_LBOX_KEY_HOOK

Description: This function is called in response to an E_CHAR event when a list box control has focus. The EVENT will always be of type E_CHAR and the WINDOW will always be of type WC_LBOX. If this attribute is set to NULL, an internal function is called that causes the selection of the first list item where the first characters of the list item match the characters of to up to 10 consecutive E_CHAR events. The comparison is case-insensitive and diacritical (accent, tilde, etc.) character-insensitive.

The following keyboard key behavior is always defined for the list box control and cannot be defined by the application list box key hook function:

Home key	Scroll to first list item
End key	Scroll to last list item
Page Up key	Scroll up one page
Page Down key	Scroll down one page
Arrow Up key	Select the item above the current selection and deselect the currently selected item
Arrow Down key	Select the item below the current selection and deselect the currently selected item

Prototype:

```
BOOLEAN XVT_CALLCONV1 lbox_key_hook_function
    (WINDOW ctlwin, EVENT * ep)
```

WINDOW ctlwin
: List box control window.

EVENT * ep
: XVT E_CHAR event record.

Uses win argument:	No
xvt_vobj_get_attr returns:	Currently set value
xvt_vobj_set_attr effect:	Sets the key callback function used by all list box controls
xvt_app_create use:	Can use either before or after
Default value	NULL

ATTR_MAC_LBOX_PROC_ID

Description: Setting this attribute to a value other than zero forces the procID to be used as the list box definition ID for new list box controls created with the native Mac Toolbox function LNew. Set the value prior to a call to xvt_ctl_create_* with a control type of WC_LBOX, and reset the value after completion of the call. The value must be greater than or equal to zero.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets list box definition ID
xvt_app_create use:	Can use either before or after
Default value:	Zero
Argument type:	short

ATTR_MAC_LOW_MEMORY_THRESHOLD

Description: XVT/Mac generates a warning dialog when available memory for the application reaches a critical low memory state. This attribute controls the threshold which XVT compares to the current available memory before generating a warning. The value set is in bytes and must be between zero and LONG_MAX. Setting the value to zero eliminates any low memory checking done by XVT. The warning message can be changed by modifying the URL STRING resource with ID MAC_STR_LOW_MEM_WARNING_ID.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets low memory warning threshold
xvt_app_create use:	Can use either before or after
Default value:	524288 bytes (512 KB)
Argument type:	long

ATTR_MAC_MENU_HOOK

Description: This function is called whenever a menu selection is made. It is called after MenuSelect and before xvt_win_dispatch_event of the E_COMMAND event. If the function returns FALSE, the menu selection is ignored and XVT performs no further processing. If the function returns TRUE, XVT processes the selection normally. To disable menu selection processing through the ATTR_MAC_MENU_HOOK function, call xvt_vobj_set_attr with NULL as the value of the attribute.

The current Macintosh WindowPtr is passed in the wp parameter. The Macintosh menu ID and Macintosh menu item ID for the selection are passed in the macID and macItem parameters. A handle to the Macintosh MenuInfo record for the selected menu is passed in the menuH parameter.

Prototype:

```
BOOLEAN XVT_CALLCONV1 menu_hook_function
    (WindowPtr wp, short macID, short macItem,
    MenuHandle menuH)
```

WindowPtr wp
: Native Macintosh window pointer for the window owning the current menubar.

short macID
: Native Macintosh menu ID.

short macItem
: Native Macintosh menu item number.

MenuHandle menuH
: Native Macintosh menu handle of selected menu.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Installs menu hook function or no hook if NULL
xvt_app_create use:	Can use either before or after
Default value:	NULL

ATTR_MAC_MOUSE_CONTROL_FOCUS

Description: Helps an application combine edit controls and text edit objects in the same window. Setting this attribute to TRUE allows E_MOUSE_DOWN events to "steal" focus from an active edit control. This enables the application to switch focus to a different text edit object, or to another control. Note that a native mouse down event in an empty window background space will cause any active edit control to lose focus, which is not standard Macintosh behavior.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets mouse click control focus state
xvt_app_create use:	Can use either before or after
Default value:	FALSE
Argument type:	BOOLEAN

ATTR_MAC_NATIVE_HTML_REFERENCE

Description: Returns the HRReference value for the underlying native HTML reference object used by a WC_HTML control.

Uses win argument:	Yes
xvt_vobj_get_attr returns:	HRReference value for WC_HTML control
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Must use after
Default value:	None
Argument type:	HRReference

ATTR_MAC_NO_GRAY_DISABLED_EDIT

Description: Allows the application to keep non-grayed disabled edit control text. By default, edit controls change all contained text to gray when the edit control becomes disabled. When you set this attribute to TRUE, all text created after that time will be displayed normally, disabled or not.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets disabled text state for edit controls
xvt_app_create use:	Can use either before or after
Default value:	FALSE
Argument type:	BOOLEAN

ATTR_MAC_NO_GRAY_MAP_COLORS

Description: By default, on monochrome monitors, XVT/Mac maps all colors to three shades of gray dithered patterns, except when you use a brush pattern other than PAT_SOLID. When you choose a non-solid brush pattern, XVT/Mac follows the Macintosh (monochrome) policy of mapping all non-white colors (in either the foreground or the background of the pattern) to black. Thus, patterns composed of non-white colored areas are rendered as regions of solid black. By setting the attribute ATTR_MAC_NO_GRAY_MAP_COLORS to TRUE, you can restrict the scope of XVT/Mac's color mapping to only the gray colors (COLOR_DKGRAY, COLOR_GRAY, and COLOR_LTGRAY). Then only gray *colors* are mapped to the gray dithered *patterns*, which in turn allows colored patterns to be drawn in such a way that they resemble patterns instead of solid black regions. This attribute can be called once for the entire application (before xvt_app_create is called) or around particular draw functions.

Uses win argument:	No
xvt_vobj_get_attr returns:	Currently set value
xvt_vobj_set_attr effect:	Determines if gray dithered pattern mapping is done on monochrome monitors
xvt_app_create use:	Can use either before or after
Default value	FALSE
Argument type	BOOLEAN

See Also: The attribute ATTR_MAC_USE_COLOR_QUICKDRAW on page A-23.

ATTR_MAC_NO_LBOX_FOCUS_BOX

Description: This attribute controls whether a list box control draws a focus indicator box around itself when it receives keyboard focus. By default, a list box control draws the box for Macintosh System 7 look-and-feel compliant behavior. Use this attribute for backward compatibility if drawing the box might disrupt an existing application's control layout in a window or dialog.

Uses win argument:	No
xvt_vobj_get_attr returns:	Currently set value
xvt_vobj_set_attr effect:	Sets the focus indicator box display state for WC_LBOX controls
xvt_app_create use:	Can use either before or after
Default value	FALSE
Argument type:	BOOLEAN

Note: You can use ATTR_MAC_NO_LBOX_FOCUS_BOX for all list boxes or for a single list box if you set it before you call the functions xvt_ctl_create or xvt_ctl_create_def.

ATTR_MAC_NO_SELECT_WINDOW

Description: Suppresses SelectWindow on native mouse down events in the content region of a window. This attribute can be used to implement "floating" windows (windows that remain in front of all other windows while that application has focus).

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Suppresses or restores SelectWindow call
xvt_app_create use:	Can use either before or after
Default value:	FALSE
Argument type:	BOOLEAN

ATTR_MAC_NO_SET_CURSOR

Description: Suppresses all calls to InitCursor and SetCursor. Enable this attribute if you want the cursor to be entirely under the control of the application (using Macintosh Toolbox calls).

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Suppresses or restores calls to InitCursor and SetCursor
xvt_app_create use:	Can use either before or after
Default value:	FALSE
Argument type:	BOOLEAN

ATTR_MAC_NO_UPDATE_MENU_BAR

Description: Prevents xvt_menu_set_tree from updating the menubar. The function xvt_menu_update should be called later by the application to "clean up" the menubar.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Suppresses or restores calls to xvt_menu_update during xvt_menu_set_tree
xvt_app_create use:	Can use either before or after
Default value:	FALSE
Argument type:	BOOLEAN

ATTR_MAC_PAT_RES_ID

Description: Allows the use of application-defined patterns (otherwise, you are limited to the 10 portable patterns defined in the XVT constant PAT_STYLE). Set the pat member of the CBRUSH structure to PAT_SPECIAL. Set this attribute to the native resource ID of a PAT# resource (PAT# is the native pattern list resource type). Then set the attribute ATTR_MAC_PAT_RES_INDEX to the index into the pattern list resource for the currently desired pattern. The value of ATTR_MAC_PAT_RES_INDEX must be a valid PAT# resource index in the pattern list resource. Note that the first pattern in the PAT# resource has an index of 1.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets the pattern list resource used
xvt_app_create use:	Can use either before or after
Default value:	sysPatListID
Argument type:	short

See Also: The attribute ATTR_MAC_PAT_RES_INDEX on page A-13.
For more information on application-defined patterns, see the "QuickDraw Drawing" chapter in the book *Inside Macintosh: Imaging with QuickDraw*.

ATTR_MAC_PAT_RES_INDEX

Description: This attribute is used only in conjunction with ATTR_MAC_PAT_RES_ID to define the index for a PAT# resource.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets the pattern list index used
xvt_app_create use:	Can use either before or after
Default value:	One
Argument type:	short

See Also: The attribute ATTR_MAC_PAT_RES_ID on page A-13.

ATTR_MAC_PIXMAP_GWORLD_DEPTH

Description: This attribute allows the application to control the color depth and thus the memory size of the GWorld used internally to represent a W_PIXMAP. The value can be set to one of the following values: 0, 1, 2, 4, 8, 16, or 32. Using lower depth values, however, may cause degradation of your W_PIXMAP image. Using a value of 0 allows the system to decide the best setting for the current monitor depth.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets the pixmap GWorld depth
xvt_app_create use:	Can use either before or after
Default value:	Zero
Argument type:	short

ATTR_MAC_PRINT_CLIPPING

Description: Certain printers do not allow the clip region to be set for the native print window GrafPort when printing. This attribute allows the application to disable print window clipping when printing to problem printers.

Uses win argument:	No
xvt_vobj_get_attr returns:	Currently set value
xvt_vobj_set_attr effect:	Sets print clipping enabled or disabled
xvt_app_create use:	Can use either before or after
Default value:	TRUE
Argument type:	BOOLEAN

ATTR_MAC_PRINT_COPIES

Description: Allows access to print job information—specifically, the number of requested copies. Set this attribute with a call to the function xvt_print_create_win. Note that some laser printer drivers handle copies at the printer level rather than at the application level. Thus, this information is not available as the data is not stored in the print job record.

Uses win argument:	No
xvt_vobj_get_attr returns:	Currently set value
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Must use after
Default value:	Zero
Argument type:	int

See Also: For more information on printing, see the "Printing Manager" chapter in the book *Inside Macintosh: Imaging with QuickDraw*.

ATTR_MAC_PRINT_FIRST_PAGE

Description: Allows access to print job information about the first page to be printed. Set this attribute with a call to the function xvt_print_create_win.

Uses win argument:	No
xvt_vobj_get_attr returns:	Currently set value
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Must use after
Default value:	Zero
Argument type:	int

See Also: For more information on printing, see the "Printing Manager" chapter in the book *Inside Macintosh: Imaging with QuickDraw*.

ATTR_MAC_PRINT_LAST_PAGE

Description: Allows access to print job information about the last page to be printed. Set this attribute with a call to the function xvt_print_create_win.

Uses win argument:	No
xvt_vobj_get_attr returns:	Currently set value
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Must use after
Default value:	Zero
Argument type:	int

See Also: For more information on printing, see the "Printing Manager" chapter in the book *Inside Macintosh: Imaging with QuickDraw*.

ATTR_MAC_PROC_ID

Description: Setting this attribute to a value other than –1 forces the procID to be used as the window definition ID for new document (not child) windows. Set the value prior to a call to xvt_win_create_* with a window type of W_DOC, W_DBL, W_MODAL, WD_MODELESS, or WD_MODAL, and reset the value after completion of the call. The value must be greater than or equal to zero, or equal to –1.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets window definition ID or default if value is –1
xvt_app_create use:	Can use either before or after
Default value:	–1
Argument type:	int

ATTR_MAC_ROUNDED_GROUPBOX

Description: Allows the application to create a group box with rounded rectangle borders instead of square borders. (This is a common look-and-feel for Macintosh applications.) Setting the attribute to TRUE causes all subsequent group box controls to have rounded rectangle borders.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets group box control rectangle border state
xvt_app_create use:	Can use either before or after
Default value:	FALSE
Argument type:	BOOLEAN

ATTR_MAC_SCROLL_THUMBTRACK

Description: Allows an application to receive the E_CONTROL events of type SC_THUMBTRACK on the Macintosh, if desired. Setting this attribute to TRUE causes these events to be sent to the application's event handler.

Caution: Be aware that the application should not call xvt_sbar_set_pos or xvt_sbar_get_pos while processing thumbtrack events on the Macintosh. (You will notice odd behavior of the scrollbar thumb if this is done.) Use the scrollbar position passed in the thumbtrack event instead.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Determines if SC_THUMBTRACK events are sent
xvt_app_create use:	Can use either before or after
Default value:	TRUE
Argument type:	BOOLEAN

ATTR_MAC_SET_TITLE_AUTO_SELECT

Description: Allows the application to specify automatic selection of the text after xvt_vobj_set_title is called for an edit control. Setting this attribute to TRUE allows the new title to be entirely selected during a call to xvt_vobj_set_title. Setting this attribute to FALSE will place the input caret at the end of the new text. This state can be set for the entire application or set around a single xvt_vobj_set_title function call.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets edit control xvt_vobj_set_title auto select state
xvt_app_create use:	Can use either before or after
Default value:	TRUE
Argument type:	BOOLEAN

ATTR_MAC_SHOW_JOB_DIALOG

Description: Disables the display of the print job dialog in the function xvt_print_create_win. Setting the attribute to FALSE after the first call to xvt_print_create_win for a batch disables the job dialog for subsequent calls to the function xvt_print_create_win. You should set the attribute to TRUE prior to the first call to xvt_print_create_win to ensure that the print job dialog is put up at least once (to initialize the print job dialog record).

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets print job dialog display state
xvt_app_create use:	Can use either before or after
Default value:	TRUE
Argument type:	BOOLEAN

Example: This example shows how to use ATTR_MAC_SHOW_JOB_DIALOG:

```
if ((printrcd = xvt_print_create((INTPTR)&SizePrintRcd))
        != NULL)
{   ...
    xvt_vobj_set_attr(NULL_WIN,
            ATTR_MAC_SHOW_JOB_DIALOG, (long)TRUE);
    for (ele = xvt_slist_get_first(filelist);
            ele != NULL;
            ele = xvt_slist_get_next(filelist,ele))
    {
        docname = (char*)xvt_slist_get_data(ele);
        if ((pwin =
            xvt_print_create_window(printrcd,docname))
                == NULL)
            break;
        xvt_vobj_set_attr(NULL_WIN,
                ATTR_MAC_SHOW_JOB_DIALOG, (long)FALSE);
        ...
        /* Fill print window with current document */
        ...
        xvt_vobj_destroy(pwin);
    }
    xvt_vobj_set_attr(NULL_WIN,
            ATTR_MAC_SHOW_JOB_DIALOG, (long)TRUE);
}
```

ATTR_MAC_STR_HELP

Description: The value of this attribute is a string that is compared to each of the buttons in the About box to determine if the help system will be invoked. The passed string is copied to an internal buffer. Maximum string length is 256 bytes.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets default string for invoking help
xvt_app_create use:	Can use either before or after
Default value:	“Help”
Argument type:	char*

Note: During xvt_app_create (only), the value of the string resource MAC_STR_HELP_ID (if created) overrides the value of ATTR_MAC_STR_HELP.

ATTR_MAC_STR_STYLE_MENU1

Description: The value of this attribute is the text used to build a Macintosh Font Style menu. The string may contain Macintosh meta-characters (font style, accelerator, etc.) used by the Macintosh menu manager. The passed string is copied to an internal buffer. Maximum string length is 256 bytes.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets string for Font Style menu items
xvt_app_create use:	Can use either before or after
Default value:	
"Plain Text/T;-(;Bold<B/B;Italic<I/I;Underline <U/U;Outline<O;Shadow<S;-(;Condense;Extend"	
Argument type:	char*

Note: During xvt_app_create (only), the value of the string resource MAC_STR_STYLEM1_ID (if created) overrides the value of ATTR_MAC_STR_STYLE_MENU1.

ATTR_MAC_STR_STYLE_MENU2

Description: The value of this attribute is a string that represents the font styles for text in a Macintosh Font Style menu. The order and choice of characters in this string should be changed if the order or contents of the Font Style menu is changed. Each character in this string value represents a font style for a corresponding menu item in the ATTR_MAC_STR_STYLE_MENU1 string. This correspondence is one-for-one—the first character represents the font for the first menu item, the second character represents the font for the second menu item, and so forth. Characters that may be defined for the ATTR_MAC_STR_MENU2 string are as follows:

Character:	**Font style:**
P	Plain text (unmodified)
B	Bold text
I	Italic text
U	Underline text

O	Outline text
S	Shadow text
C	Condensed text
E	Extended text
<space>	Separator or unsupported font modifier

No other characters are valid. The passed string is copied to an internal buffer. Maximum string length is 256 bytes.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets string for matching Font Style menu items
xvt_app_create use:	Can use either before or after
Default value:	"P BIUOS CE"
Argument type:	char*

Note: During xvt_app_create (only), the value of the string resource MAC_STR_STYLEM2_ID (if created) overrides the value of ATTR_MAC_STR_STYLE_MENU2.

ATTR_MAC_STR_STYLE_MENU3

Description: The value of this attribute is a string that lists the font point sizes in a Macintosh Font Style menu. XVT/Mac assumes the numeric value is the first part of each item (for example, the numeral "9" in "9 point"). The passed string is copied to an internal buffer. Maximum string length is 256 bytes.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets string for Font Style menu point size items
xvt_app_create use:	Can use either before or after
Default value:	

```
"-(;9 point;10 point;12 point;14 point;
   18 point;24 point;36 point;48 point;
   72 point;Other %s%d%s..."
```

Argument type:	char*

Macintosh replaces the %s format characters in the string with parentheses and the %d with the name of the current font.

Note: During xvt_app_create (only), the value of the string resource MAC_STR_STYLEM3_ID (if created) overrides the value of ATTR_MAC_STR_STYLE_MENU3.

ATTR_MAC_SYSTEM_INITIALIZATION

Description: This function is called during xvt_app_create to initialize the native Macintosh Toolbox if the attribute is set to a value other than NULL. To see how XVT/Mac uses the initialization function, look at the file **minit.c** in the **samples:ptk:mac:** folder. Use the code in this file as a template for your own custom function. Do not remove any of the Macintosh Toolbox initialization functions in the file, as XVT/Mac depends on them. However, you can add other Toolbox initialization functions as required by your application, or modify arguments to the existing Toolbox initialization functions.

Your application can call your initialization function directly prior to xvt_app_create. However, you must ensure that your function is protected against multiple calls. Examine the internal function in **minit.c**.

Prototype: void XVT_CALLCONV1 system_initialization_function(void)

Uses win argument:	No
xvt_vobj_get_attr returns:	Currently set value
xvt_vobj_set_attr effect:	Sets the system initialization function used by xvt_app_create
xvt_app_create use:	Must use before
Default value:	NULL

ATTR_MAC_USE_COLOR_QUICKDRAW

Description: On a Macintosh with color display, XVT/Mac switches to its monochrome algorithms when the color depth is set to two. These are the same algorithms XVT uses on Macintoshes without Color QuickDraw. This attribute lets you test your programs without running on a monochrome-only Macintosh. When this attribute is set to TRUE, grays and colors (unless ATTR_NO_GRAY_MAP_COLORS is set to TRUE) are drawn with patterns instead of with a solid pattern and a gray color. You can force a Color QuickDraw-equipped Macintosh to use the color algorithms at all times—even when the color depth is two—by setting this attribute to TRUE. This attribute has no effect on Macintoshes without Color QuickDraw.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Sets QuickDraw color algorithm state
xvt_app_create use:	Can use either before or after
Default value:	FALSE
Argument type:	BOOLEAN

See Also: The attribute ATTR_NO_GRAY_MAP_COLORS on page A-10.

ATTR_MAC_USE_NATIVE_ORIGIN

Description: To make applications more portable, all windows and dialogs are created with a screen origin of 0,0 at the bottom of the menubar and extreme left of the screen. Setting this attribute to TRUE changes the origin to the top of the menubar (extreme top and left of screen). This is the native system origin.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Places the screen origin
xvt_app_create use:	Can use either before or after
Default value:	FALSE
Argument type:	BOOLEAN

ATTR_MAC_WIN_MAX_HEIGHT
ATTR_MAC_WIN_MAX_WIDTH
ATTR_MAC_WIN_MIN_HEIGHT
ATTR_MAC_WIN_MIN_WIDTH

Description: Setting these attributes defines the minimum or maximum height or width of the client rectangle in which a user can resize a window. These attributes only affect windows that have a size box and have no effect on window resizing with xvt_vobj_move.

Uses win argument:	Yes
xvt_vobj_get_attr returns:	Previously set value for win
xvt_vobj_set_attr effect:	Sets the minimum or maximum height or width for win
xvt_app_create use:	Must use after
Default value:	SHRT_MAX for maximums, 80 for minimums
Argument type:	short

ATTR_MAC_WIN_USE_FIRST_CLICK

Description: Setting this attribute to TRUE for a given window allows that window to receive E_MOUSE_DOWN or E_CONTROL events, in addition to the E_FOCUS event, when a mouse click causes the window to become active. You can set this attribute for top-level and child windows, but not for dialogs or controls. Because this behavior is not standard Macintosh look-and-feel compliant, the attribute should be used only to implement special features. For example, if you want to include a floating toolbar window in your application, you could set this attribute to TRUE because the window is always fully visible (or fully invisible). Thus, the user's click in the toolbar's controls has no adverse affect, since the controls in the floating toolbar will never be obscured by other windows.

Uses win argument:	Yes
xvt_vobj_get_attr returns:	Previously set value for win
xvt_vobj_set_attr effect:	Sets first click state for win
xvt_app_create use:	Must use after
Default value:	FALSE
Argument type:	BOOLEAN

A.2. Variations on Portable Attributes

These portable attributes have been modified slightly to support differences on the native Macintosh platform.

ATTR_EVENT_HOOK

Description: A pointer to a hook function that is called whenever a native Macintosh event is generated for a window or dialog in your application. (It is called after WaitNextEvent and before xvt_win_dispatch_event.) The Mac EventRecord parameter pevent is a pointer to the event record retrieved from the Macintosh system event queue. An XVT WINDOW is passed as the win parameter if it is appropriate for the event. If no window is appropriate or if the window is not an XVT WINDOW, win is passed as a value of NULL_WIN. The BOOLEAN parameter ismodal is set to TRUE if the window is a modal dialog.

Your application can process this message data in any appropriate manner—however, modifying this data may have undesired side effects in subsequent processing of the event by XVT/Mac. If your hook function returns FALSE, XVT does not process the event further. If your hook function returns TRUE, XVT processes the event normally.

EventRecord is defined as follows in the file **Events.h**:

```
struct EventRecord{
        short what;
        long message;
        long when;
        Point where;
        short modifiers;
};

typedef struct EventRecord EventRecord;
```

Prototype:

```
BOOLEAN XVT_CALLCONV1 event_hook_function
    (WINDOW win, BOOLEAN ismodal, EventRecord * pevent)
```

WINDOW win
: Window receiving event.

BOOLEAN ismodal
: Window is a modal dialog.

EventRecord * pevent
: Native Macintosh event.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Installs hook or uninstalls hook if value is NULL
xvt_app_create use:	Can use either before or after
Default value:	NULL

ATTR_HAVE_COLOR

Description: A BOOLEAN value indicating whether the application is running on a color system. Set to TRUE during xvt_app_create if the main display is a color monitor.

Uses win argument:	No
xvt_vobj_get_attr returns:	Color monitor state
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Must use after
Default value:	FALSE
Argument type:	BOOLEAN

ATTR_HAVE_MOUSE

Description: A BOOLEAN value indicating whether or not the program is running on a system with a mouse or other pointing device present. Note that this attribute is always TRUE on Macintosh, because there is no way to query the system if the mouse is present or not.

Uses win argument:	No
xvt_vobj_get_attr returns:	Always TRUE
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Must use after
Default value:	TRUE
Argument type:	BOOLEAN

ATTR_KEY_HOOK

Non Multibyte-aware Application

If your application uses a single-byte character codeset and you have set the value of ATTR_MULTIBYTE_AWARE as FALSE (default), then ATTR_KEY_HOOK behaves as follows:

Description: A pointer to a hook function that is called *after* native keyDown or autoKey events are received and *before* E_CHAR events are sent to your application. The keycode, modifiers, and chrp parameters are copies of data retrieved from the Macintosh system event queue. An XVT WINDOW is passed as the win parameter if it is appropriate for the event. If no window is appropriate or if the window is not an XVT WINDOW, win is passed with a value of NULL_WIN. The BOOLEAN parameter ismodal is set to TRUE if the window is a modal dialog.

If you need to perform key translation, you must modify data in the modifiers, if appropriate, and chrp parameters. XVT uses these parameters to construct an E_CHAR event. If your hook function returns FALSE, XVT dispatches the E_CHAR event without further processing. If your hook function returns TRUE, XVT processes the event normally and dispatches the E_CHAR event to your application XVT event handler.

Prototype:
```
BOOLEAN XVT_CALLCONV1 key_hook(WINDOW win,
    BOOLEAN ismodal, short * keycode, short * modifiers,
    short * chrp)
```

WINDOW win
: Window receiving key event.

BOOLEAN ismodal
: Window is a modal dialog.

short * keycode
: Native Macintosh keycode value.

short * modifiers
: Native Macintosh key modifiers flags.

short * chrp
: Native Macintosh character value.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Installs hook function, or reinstalls default hook if value is NULL
xvt_app_create use	Can use either before or after
Default value:	user_key_hook found in **mhook.txt** (**samples:ptk:mac:** folder)

Multibyte-aware Application

If your application is multibyte-aware (in other words, you have set the value of ATTR_MULTIBYTE_AWARE as TRUE), then ATTR_KEY_HOOK behaves as follows:

Description: A pointer to a hook function that is called *after* native keyDown or autoKey events are received and *before* E_CHAR events are sent to your application. The keycode and modifiers parameters are copies of data retrieved from the Macintosh system event queue. The ep parameter is a pointer to the E_CHAR event structure. An XVT WINDOW is passed as the win parameter if it is appropriate for the event. If no window is appropriate or if the window is not an XVT WINDOW, win is passed with a value of NULL_WIN. The BOOLEAN parameter ismodal is set to TRUE if the window is a modal dialog.

If you need to perform key translation, you must modify data in the ep parameter. XVT will dispatch the E_CHAR event returned. If your key hook function translates a character to a virtual key, then it should also set the event ep->v.chr.virtual_key field to TRUE. If your hook function returns FALSE, XVT dispatches the E_CHAR event without further processing. If your hook function returns TRUE, XVT processes the event normally and dispatches the E_CHAR event to your application XVT event handler.

Prototype:
```
BOOLEAN XVT_CALLCONV1 key_hook(WINDOW win,
    BOOLEAN ismodal, short * keycode, short * modifiers,
    EVENT * ep);
```

WINDOW win
: Window receiving key event.

BOOLEAN ismodal
: Window is a modal dialog.

short * keycode
: Native Macintosh keycode value.

short * modifiers
: Native Macintosh key modifiers flags.

EVENT * ep
: XVT character event.

Uses win argument:	No
xvt_vobj_get_attr returns:	Previously set value
xvt_vobj_set_attr effect:	Installs hook function, or reinstalls default hook if value is NULL
xvt_app_create use	Can use either before or after
Default value:	user_key_hook found in **mhook.txt** (**samples:ptk:mac:** folder)

ATTR_NATIVE_GRAPHIC_CONTEXT

Description: A value that represents the underlying graphics context used by the native window system for a particular window. Returns the Macintosh CGrafPtr for an XVT window. The window must be a valid XVT WINDOW that is not a control. Note that child windows return the CGrafPtr for their root level parent. You must include the Macintosh Toolbox include file **QuickDraw.h** in order to have the Macintosh type CGrafPtr defined.

Uses win argument:	Yes
xvt_vobj_get_attr returns:	Macintosh CGrafPtr value for WINDOW
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Must use after
Default value:	None
Argument type:	CGrafPtr

ATTR_NATIVE_WINDOW

Description: A value that represents the underlying window object used by the native window system, for a particular window. Returns the Macintosh WindowRef for an XVT window. The window must be a valid XVT WINDOW that is not a control. Note that child windows return the WindowRef for their root level parent. You must include the Macintosh Toolbox include file **QuickDraw.h** in order to have the Macintosh types WindowRef defined.

Uses win argument:	Yes
xvt_vobj_get_attr returns:	WindowRef value for WINDOW
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	No
Default value:	None
Argument type:	WindowRef

ATTR_NUM_TIMERS

Description: The number of timers in a system available to the application via xvt_timer_create. On the Macintosh, this attribute always returns SHRT_MAX because the number of timers is limited only by available memory.

Uses win argument:	No
xvt_vobj_get_attr returns:	Always SHRT_MAX
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Must use after
Default value:	SHRT_MAX
Argument type:	short

ATTR_PRINTER_*

Description: The ATTR_PRINTER_* attributes do not report accurate information on the Macintosh because xvt_vobj_get_attr doesn't have access to the current print record. Instead, you should use the escape function XVT_ESC_GET_PRINTER_INFO.

Uses win argument:	No
xvt_vobj_get_attr returns:	Values for default print record
xvt_vobj_set_attr effect:	Illegal
xvt_app_create use:	Must use after
Default value:	Values for default print record
Argument type:	long

A.3. Non-Portable Escape Codes

The xvt_app_escape function enables you to set or get XVT/Mac-specific information that you cannot set or get using the xvt_vobj_set_attr or xvt_vobj_get_attr functions. The xvt_app_escape function's escape codes and the associated parameter lists are given below, with a brief explanation of types and values. The escape code is an integer whose value is defined internally by XVT.

XVT_ESC_MAC_DIALOG_POSITION

Description: This escape determines the proper screen position for a dialog according to Apple's *Macintosh Human Interface Guidelines* on Dialog Positions. To use this function, make the application's dialogs invisible in XVT-Design or in URL. In the E_CREATE for the dialog event handler, call this function to get a corrected client rect, then call xvt_vobj_move followed by xvt_vobj_set_visible on the dialog window. The passed rect does not need to be initialized.

Prototype:
```
xvt_app_escape(XVT_ESC_MAC_DIALOG_POSITION,
    WINDOW win, RCT * prct);
```

WINDOW win
: Dialog window.

RCT * prct
: Dialog-modified screen rectangle.

XVT_ESC_MAC_FONT_GET_RES_NAME

Description: Given a native Macintosh font resource ID, this function returns the font family resource name. The application must allocate memory for the returned name before calling this function. The size of the allocated space is passed in maxsize.

Prototype:
```
xvt_app_escape(XVT_ESC_MAC_FONT_GET_RES_NAME,
    short macfontid, char * pname, short maxsize);
```

short macfontid
: Macintosh font resource ID.

char * pname
: Returned Macintosh font resource name.

short maxsize
: Maximum length of name.

XVT_ESC_MAC_GET_DESKTOP_BOUNDS

Description: This escape calculates the bounding rectangle of the desktop's current visible region. This rectangle potentially spans multiple monitors. On systems with more than one monitor, the physical layout of the desktop is defined in the Monitors control panel.

Prototype:
```
xvt_app_escape(XVT_ESC_MAC_GET_DESKTOP_BOUNDS,
    RCT * bounds);
```

RCT * bounds
: Desktop bounding rectangle returned.

XVT_ESC_MAC_GET_DISPLAY_INFO

Description: This escape calculates the bounding rectangle of the specified display within the desktop's current visible region, and specifies color and pixel size. Displays are specified by index into the native graphics device list. Index 0 refers to the main display, i.e., the display containing the menubar. xvt_app_escape returns FALSE if the index does not correspond to an existing display.

Prototype:
```
xvt_app_escape(XVT_ESC_MAC_GET_DISPLAY_SIZE,
    short index, RCT * bounds, BOOLEAN * is_color,
    short * psize);
```

short index
: Display index.

RCT * bounds
: Returned bounding rectangle of display.

BOOLEAN * is_color
Color = TRUE, Grayscale = FALSE.

short * psize
Returned pixel size for display.

XVT_ESC_MAC_GET_EDIT_HANDLE

Description: This escape returns the text edit handle for a WC_EDIT or WC_LISTEDIT control.

Prototype:
```
xvt_app_escape(XVT_ESC_MAC_GET_EDIT_HANDLE, WINDOW win,
    TEHandle * ptehandle);
```

WINDOW win
Control window for a WC_EDIT or WC_LISTEDIT control.

TEHandle * ptehandle
Returned text edit handle for the control.

XVT_ESC_MAC_GET_LIST_HANDLE

Description: This escape returns the list handle and other information for a WC_LBOX control.

Prototype:
```
xvt_app_escape(XVT_ESC_MAC_GET_LIST_HANDLE, WINDOW win,
    ListHandle * plist_handle, Rect * pview, int * pfont,
    int * psize);
```

WINDOW win
Control window for a WC_LBOX control.

ListHandle * plist_handle
Returned list handle for the control.

Rect * pview
Returned viewing rectangle for the list box.

int * pfont
Returned Macintosh font number set for the list box.

int * psize
Returned font size set, in points, for the list box.

XVT_ESC_MAC_GET_PICT_ID

Description: Given an XVT control window for a picture or icon control, this function returns the native Macintosh resource ID of the PICT or ICON resource used by the control. A picture control is created by making a dialog item of type picture in the native resources.

Prototype:
```
xvt_app_escape(XVT_ESC_MAC_GET_PICT_ID, WINDOW win,
    long * pid);
```

WINDOW win
: XVT picture or icon control window.

long * pid
: Returned Macintosh PICT resource ID for picture control or ICON resource ID for icon control.

XVT_ESC_MAC_MODAL_WINDOW

Description: This escape makes an XVT window modal by limiting mouse clicks to the window and disabling the menubar; it works only on windows of type W_DBL or W_DOC. This escape could be useful to a C++ programmer because it offers a separation between the modality of a window and the creation of a window. Another indirect benefit is that W_DBL and W_DOC windows can use different border styles than W_MODAL windows, which might be useful in certain situations. However, remember that choosing border styles other than the style provided by W_MODAL windows changes the look-and-feel of the application, and takes the application away from the standard Macintosh look-and-feel. The function returns when the window is closed.

Prototype:
```
xvt_app_escape(XVT_ESC_MAC_MODAL_WINDOW, WINDOW win);
```

WINDOW win
: W_DBL or W_DOC type window.

XVT_ESC_MAC_PALET_GET_PALETTE_HANDLE

Description: Given an XVT_PALETTE, this function returns the corresponding native PaletteHandle.

Prototype: xvt_app_escape(XVT_ESC_MAC_PALET_GET_PALETTE_HANDLE, XVT_PALETTE palet, PaletteHandle * pmacpalette);

XVT_PALETTE palet
: XVT_PALETTE to get native palette from.

PaletteHandle * pmacpalette
: Returned native PaletteHandle. If an error occurs, the handle will be set to NULL.

XVT_ESC_MAC_PICT_READ_FROM_FILE

Description: Given a file pointer specifying a valid Macintosh PICT file, this function will read the contents and return a PICTURE and the PICTURE's bounding rectangle.

Prototype: xvt_app_escape(XVT_ESC_MAC_PICT_READ_FROM_FILE, FILE * filep, PICTURE * pict, RCT * rctp);

FILE * filep
: Macintosh PICT file.

PICTURE * pict
: Returned PICTURE representing the Macintosh PICT that was read. If an error occurs during reading, NULL_PICT will be returned.

RCT * rctp
: Returned bounding rectangle for the XVT PICTURE.

XVT_ESC_MAC_PICTURE_COMMENT

Description: This escape allows the application to access the native Macintosh Toolbox function PicComment. Calling native Macintosh Toolbox functions while in the context of an open PICTURE (for a particular window) does not append those function calls to the PICTURE object.

Prototype: xvt_app_escape(XVT_ESC_MAC_PICTURE_COMMENT, WINDOW win, short kind, short datasize, Handle datahandle);

WINDOW win
: XVT WINDOW with an open PICTURE.

short kind
: The kind parameter passed to PicComment.

short datasize
The datasize parameter passed to PicComment.

Handle datahandle
The datahandle parameter passed to PicComment.

XVT_ESC_MAC_RES_GET_PICT

Description: Given an ID for a PICT resource, this function reads the PICT from the current resource list and returns a PICTURE and the PICTURE's bounding rectangle.

Prototype:
```
xvt_app_escape(XVT_ESC_MAC_RES_GET_PICT, short id,
    PICTURE * pict, RCT * rctp);
```

short id
Resource ID of PICT.

PICTURE * pict
Returned PICTURE representing the Macintosh PICT that was read. If an error occurs during reading, NULL_PICT will be returned.

RCT * rctp
Returned bounding rectangle for the XVT PICTURE.

XVT_ESC_MAC_SET_PICT_ID

Description: Given an XVT control window for a picture control or an icon control, this function sets the native Macintosh PICT or ICON resource ID used by the control. A picture control is created by defining a dialog item of type picture in the native resources.

Prototype:
```
xvt_app_escape(XVT_ESC_MAC_SET_PICT_ID, WINDOW win,
    long id);
```

WINDOW win
XVT picture or icon control window.

long id
New Macintosh PICT resource ID for picture control or ICON resource ID for icon control.

XVT_ESC_MAC_SET_WINDOW_COLOR

Description: This escape allows the application to set the standard color table for individual windows or dialogs. The function takes five pointers to Macintosh RGBColor structures that define: window content color, frame color, text color, highlight color, and title bar color. Any of the color arguments can be NULL, in which case, the color uses the default color for that field.

When using this function, create the window or dialog as invisible and call xvt_vobj_set_visible after calling this escape function. This prevents the window or dialog from displaying with the default colors, and then changing when the application calls xvt_app_escape.The structure RGBColor is defined in the Macintosh include file **QuickDraw.h**, so you must include this file.

Prototype:

```
xvt_app_escape(XVT_ESC_MAC_SET_WINDOW_COLOR,
    WINDOW win, RGBColor * pcontextColor,
    RGBColor * pframeColor, RGBColor * ptextColor,
    RGBColor * philiteColor, RGBColor * ptitleBarColor);
```

WINDOW win
: XVT window or dialog for which the color is being set.

RGBColor * pcontextColor
: Color for the window or dialog context or background.

RGBColor * pframeColor
: Color for the window or dialog frame.

RGBColor * ptextColor
: Color for the window or dialog title text.

RGBColor * philiteColor
: Color for the window or dialog active window highlight.

RGBColor * ptitleBarColor
: Color for the window or dialog title bar.

B

APPENDIX B: FREQUENTLY ASKED QUESTIONS

Q: *Why do I get the XVT Fatal Error, "Failure initializing resources, Resource fork missing?, File:mres.c, Line 32," or an error with ID 0x00509C42 when I attempt to run my program?*

A: This error condition usually occurs when XVT/Mac can't locate the resources that should have been attached to the application during the build process.

xrc must be used to convert the **.url** file to a text-based native resource file. The text-based native resource file must have the extension **.r** and must be added as a source file to the compiler project.

Q: *When running **xrc**, why do I get an error message indicating that the **xvt.h** or **url.h** include file cannot be found?*

A: To find the necessary include files, **xrc** relies on a small text file, **xrc.opt**, found in the same folder as **xrc**. For a number of reasons, this file may not provide the correct path. The best solution is to edit the file using any text editor, inserting a complete, correct pathname. Be sure to enclose your defined include path in double quotes (" ").

Example: This example shows a typical include path (note that each folder is separated by a colon):

```
-i "HD:xvtdsc56:mac_ppc:include:"
```

Q: *I get an error from XVT but I can't read it. It appears to be some hex-coded number.*

A: This is an internal code that the XVT error handler uses to search the **ERRCODES.TXT** file. However, your application, when it displays an error, checks only the current directory for this file. Place a copy of **ERRCODES.TXT** in the same folder as your application. (**ERRCODES.TXT** can be found in the **doc** folder.) Make sure that you *copy* the original rather than move it. When the error is displayed, you will get a text string rather than a code. Alternatively, you can set the attribute ATTR_ERRMSG_FILENAME to the pathname of the file.

Q: *How do I interpret the error message screen?*

A: The four lines of an error are:

- The first line states the level of the error (WARNING, ERROR, or FATAL) as well as the message text such as "Invalid rectangle passed into xvt_…".
- The second line states the category of the error and a description of the category. The category information comes from the file **xvt_msgs.h**, located in your include folder.
- The third line displays a stack trace, starting from xvt_app_create, listing the known XVT functions from left to right. The last known called XVT function is listed on the right. This call is the source of the warning, error, or fatal error. The stack trace does *not* list native functions or your own function calls. By default, it only recognizes XVT functions.
- The last line shows the source code file where the error originated as well as the line number of the xvt_errmsg_sig call.

Q: *The error message says that I have an error in a particular function call but I know for a fact that I never call that function in my code (or my code is calling it correctly). What's wrong?*

A: You are probably correct—the source of the error may not be from your code calling the function, but rather, from another XVT function that calls the function. To trace the actual source of the problem, you can install the following error handler:

```
static BOOLEAN                  XVT_CALLCONV1
ErrHandler                      XVT_CALLCONV2
XVT_CC_ARGL((err,context))
XVT_CC_ARG (XVT_ERRMSG, err)
XVT_CC_LARG (DATA_PTR, context)
{
        NOREF(err);
        NOREF(context);
        return FALSE;  /* did not handle this error, */
                                /* pass it along */
}
```

Install this error handler either prior to or after xvt_app_create as follows:

```
/* Establish a permanent error handler */
xvt_vobj_set_attr (NULL_WIN, ATTR_ERRMSG_HANDLER,
        (long)ErrHandler);
```

Set a breakpoint at ErrHandler and run your program in the debugger. You'll be able to see where your code is generating this message and more clearly discern the cause.

Q: *How can I override the About box?*

A: There are two ways to do this. If you don't want to add to the controls in the box but want to modify the look (slightly), look in the **url_plat.h** for the URL code describing the About box. You can delete controls from this dialog but you cannot add to them! XVT does not recommend deleting controls from the About box.

The second (and better) way to override the About box is to create your own menu hook function that you call from ATTR_MAC_MENU_HOOK. In the prototype for this function, only use the second and third parameters (macID and macItem)—you can ignore the rest. XVT's internal representation for the Apple menu is macID = 32767. The first item on this menu, the About box item, is macItem = 1. Make sure your menu hook function returns FALSE if this item is selected and returns TRUE for all others. You can test for these values in the menu hook function and call your own dialog or window creation function (i.e., xvt_dlg_create_res). Make sure that the container (your About box window) is modal; one approach is to make it a modal window. You will then have full access to the event handler to draw graphics, display QuickTime movies, etc.

Q: *If I fill my list box past a certain limit on the Macintosh, it starts to behave strangely. I don't have this problem on other platforms. What's wrong?*

A: List boxes on XVT/Mac have a 32KB limit for the data you can store in them. If you need to scroll through a large information list, create a window that you can scroll through on your own. The native Macintosh List Manager was not designed to handle data beyond 32KB.

Q: *On the Macintosh, my string resources aren't accessible with xvt_res_get_str. They seem to be accessible only with xvt_res_get_str_list. How can I get them using xvt_res_get_str?*

A: **xrc** generates two native types of string resources for the Macintosh: STR and STR#. The former is created for an isolated string resource listed in your URL file. Its resource ID does not directly follow or precede any other string resource ID. The latter, STR#, comes from lists of strings which are shown in the URL file by a series of consecutive numbers. xvt_res_get_str looks for STR type resources. xvt_res_get_str_list looks for STR# type resources.

If you don't want to use the string lists, you can edit your string lists to utilize non-consecutive resource IDs, or simply place the resource IDs in reverse order. Either technique forces these resources to show up in the Rez file as STR rather than STR#.

Q: *When compiling my application with the XVT PTK 4.x release, I sometimes get the following XVT internal warning:*

```
WARNING: API function already marked in frame
Category: Error messaging facility (Error Message Frame problems)
Function: xvt_app_process_pending_events
xvt_app_process_pending_events
File: ./verrmsg.c line: 405
```

What does this mean and how can I correct the problem?

A: With error handling in XVT PTK 4.x, each time a function call is made, it is "marked." When it returns, it is "unmarked," so that XVT can report which call caused the error.

Since 4.x was released, we have learned that the marking and unmarking of function calls does not happen correctly in certain cases, particularly, in cases where the application interacts with the toolkit directly in such a way that causes recursion. Thus, the

warning occurs. The warning is harmless and should not affect the operation of the application at all.

To prevent the message, however, you can override the error message handler by creating one that filters out the warning message. You can install such a message handler from the window event handler for C customers, as follows:

```
static BOOLEAN                  XVT_CALLCONV1
ErrHandler                      XVT_CALLCONV2
#ifdef XVT_CC_PROTO
(
XVT_ERRMSG   err,               /* Error Message Object */
DATA_PTR     context            /* Context, (not used here) */
)
#else
( err, context )
XVT_ERRMSG   err;
DATA_PTR     context;
#endif
{
       /* Check for error signal(s) we want to ignore */
             if (xvt_errmsg_get_msg_id(err) ==
             ERR_EMF_FRAME_MARKED)
      return TRUE;     /* forget this message,
                  it's OK */
       /* Pass the remaining signals to the default
             handler */
   return FALSE;
}
```

You can also override the warning in the task window event handler or prior to calling xvt_app_create do, as follows:

```
case E_CREATE:
xvt_vobj_set_attr (win, ATTR_ERRMSG_HANDLER,
       (long)ErrHandler);
```

If you are using C++, you can install an error handler by placing the following line in header file:

```
BOOLEAN ErrHandler(XVT_ERRMSG err, DATA_PTR context);
```

Place the following line in an implementation file after the #include statements:

```
extern BOOLEAN ErrHandler(XVT_ERRMSG err,
       DATA_PTR context);
```

Place the following function definition in the implementation file:

```
BOOLEAN             XVT_CALLCONV1
ErrHandler          XVT_CALLCONV2
#ifdef XVT_CC_PROTO

(
XVT_ERRMSG   err,           /* Error Message Object */
DATA_PTR     context        /* Context, (not used here) */
)
#else
( err, context )
XVT_ERRMSG   err;
DATA_PTR     context;
#endif
{
   /* Check for error signal(s) we want to ignore */
   if (xvt_errmsg_get_msg_id(err) == ERR_EMF_FRAME_MARKED)
      return TRUE;    /* forget this message, it's OK */

   /* Pass the remaining signals to the default
            handler */
   return FALSE;
}
```

Then place the following line in **cstartup.cxx** after the #includes (You need to have a header file with the function prototype included):

```
extern BOOLEAN ErrHandler(XVT_ERRMSG err,
      DATA_PTR context);
```

Finally, in CApplication() theApplication:

```
CApplication0 theApplication;
// add the following ATTR statement here
xvt_vobj_set_attr (NULL_WIN, ATTR_ERRMSG_HANDLER,
      (long)ErrHandler);
theApplication.Go(
```

In summary, install the event handler so that it ignores the warning message.

Q: *How do I use color with controls in my application?*

A: You can use the following two Portability Toolkit functions to set colors for controls in your application:

```
void xvt_ctl_set_colors(WINDOW ctl_win,
      // WINDOW ID of the control
XVT_COLOR_COMPONENT *colors,
      // colors to set or unset
XVT_COLOR_ACTION action)
      // set or unset the colors
```

and

```
void xvt_win_set_ctl_colors(WINDOW win,
      // WINDOW ID of the window or dialog
XVT_COLOR_COMPONENT *colors,
      // colors to set or unset
XVT_COLOR_ACTION action)
      // set or unset the colors
```

xvt_ctl_set_colors sets or unsets the colors for a single control. This function overrides any color values you set previously for the control, but only for the XVT_COLOR_COMPONENT of the colors array. All other colors used by the specified control are not affected. To set the default colors for a control, use NULL for the value of colors. An action value of XVT_COLOR_ACTION_SET sets the control colors for the color components specified in the colors parameter. An action value of XVT_COLOR_ACTION_UNSET sets the control colors for the color components specified in the colors parameter to colors inherited from the control's container, the colors owned by the application, or the system default.

xvt_win_set_ctl_colors sets or unsets the colors for all existing controls in window win and all controls that you create after setting the colors. It will not change the colors of controls in other windows. This function overrides any color values you set previously for the controls in the window, but only for the XVT_COLOR_COMPONENT of the colors array. All other colors used by the window's control are not affected.

Note: For controls with color components set individually, the components that were set *will not* be affected by this color change. The components that *were not* set *will* be affected. For example, if a pushbutton has blue set for the foreground color and the window has red set for the background color, the background of the pushbutton will be red.

To set the default colors for controls in a window, use NULL for the value of colors. XVT_COLOR_ACTION_SET and XVT_COLOR_ACTION_UNSET work as described above. Note that this function does not affect the colors of the container decorations or any other colors that appear in the container itself.

The following Portability Toolkit functions allow you to get the currently-defined color settings:

```
XVT_COLOR_COMPONENT *xvt_ctl_get_colors(
      WINDOW ctl_win)
```

and

```
XVT_COLOR_COMPONENT *xvt_win_get_ctl_colors(
      WINDOW win)
```

Q: *Where are all new features of the PTK documented?*

A: New functionality is outlined in the *XVT Portability Toolkit Reference* and in the *XVT Portability Toolkit Guide*, both of which you will find on the documentation CD. XVT has chosen to use an online format to make reference information clearer, easier to find, and more usable.

In addition to documenting new functionality, the online *XVT Portability Toolkit Reference* contains sections on each of the following topics:

- XVT Portable Attributes
- XVT Events
- XVT Data Types
- XVT Constants
- XVT Functions
- URL Statements
- Help File Statements
- Tools

Q: *How do standard fonts map to multibyte fonts?*

A: XVT does not automatically map to multibyte fonts. In order for your application to use multibyte fonts, you must first Internationalize and Localize your application, using the methods detailed in Chapter 19 of the *XVT Portability Toolkit Guide*. You must also install the multibyte fonts appropriate for the language you intend to use, according to your system guidelines. This will allow the fonts to be available to your XVT application.

Presumably, you will be translating your application to one or more languages. If you have properly internationalized your application, all your text and font references exist only in your resource file. When you translate your text, you should also setup the font and font_map resource appropriate for each language.

To set a multibyte font, you must modify the URL font or font_map statements of your application to contain native fonts appropriate for the language.

XVT supplies the following LANG_* xrc compiler options (files in your **ptk/include** directory):

- LANG_JPN_SJIS supports Japanese in Shift-JIS code (file **ujapsjis.h**)
- LANG_GER_IS1 supports German in ISO Latin 1
- LANG_GER_W52 supports German in Windows 1252
- Files for English, French, and Italian are also provided

These options and others are listed and discussed further in the *XVT Portability Toolkit Guide* and the *Guide to XVT Development Solution for C++*.

XVT cannot guarantee which character set your customers will use. There is more than one set available for many languages. Because the font to which you map must be available on your customer's system in order for your application to run, a survey of your proposed customer base may be in order.

The availability of these fonts and other system setup issues should become part of the installation requirements for your application, or the fonts should be installed with your application.

Q: *I've completed development and thoroughly tested my application. I understand the XVT Portability Toolkit has compile time optimization. How do I enable it?*

A: In order to understand how XVT compile time optimization works, some knowledge of the XVT Portability Toolkit implementation is required. The XVT Portability Toolkit is implemented in two layers. The top API layer, the functions of which are listed in the *XVT Portability Toolkit Reference*, is called directly by your application. This layer performs error checking of all input parameters and sometimes other validation before calling the internal layer. It is the internal layer that contains the implementation of the functionality.

XVT provides a compile time symbol, XVT_OPT, which, when defined during application compilation, redefines the top level function names to directly call the internal API functions through macros. This bypasses the parameter checking provided by the top layer and eliminates an extra stack level for each XVT API function. You can also leave XVT_OPT undefined, allowing for the specific optimization of your application code. The header file **xvt_opt.h** contains the macro definitions of the XVT API functions that are optimized.

The optimization will not eliminate all error checking from the XVT Portability Toolkit. Rather, it will eliminate only those errors related

to XVT API function parameters. Also, because the top layer sets up the error frames for function information, any errors that do occur may have fictitious results for the function stack trace.

XVT recommends this option be used only after you have completed development and have thoroughly tested your application. Attempting to use this option too early in your development process may result in application crashes and other odd behavior due to improperly called functions that would otherwise have been checked and diagnosed by the top API layer.

Q: *How are the text edit objects after version 4.5 different from the text-edit objects in previous versions?*

A: Text-edit controls have been enhanced to work more like other objects in the XVT Portability Toolkit. The text edits after 4.5 have two improvements over those in previous versions. First, they have been placed inside a child window. Second, you can now use some of the same routines to manipulate controls and text-edits. Additionally, if you find you still need the previous types of text-edits, you can continue to use them.

In releases after 4.5, text-edits have been placed inside a child window, ensuring that they have more consistent behavior with other controls. For instance, the insertion point that appears in editable controls now acts more consistently. You can be assured that only one control will possess the insertion point at any one time. Also, to maintain backward compatibility, the previous text-edit functions will still work. That is, you may continue to use text-edit controls that are not contained in a child window. To use old text-edit features, read about using the attribute ATTR_R40_TXEDIT_BEHAVIOR in the *XVT Portability Toolkit Guide*.

Since the old-style text-edit is fundamentally different from other controls, it required specialized text-edit functions. However, you can manipulate the new text-edit features with many of the generic control and window functions. For instance, with the old-style text-edit, code would have to decide whether a text-edit or some other control should receive focus, requiring the use of two functions, xvt_tx_set_active() and xvt_scr_set_focus_vobj(). The new text-edit lets you use xvt_scr_set_focus_vobj() to clean up some code. xvt_scr_set_focus_vobj() can be helpful if an application needs to handle arrays of native controls mixed with text-edits.

Note: Not all the ctl functions work on text-edits. Check the *XVT Portability Toolkit Guide* for specific functions you can use.

Q: *I'm not sure I understand the M_* values for DRAW_MODE as stated in the XVT Portability Toolkit Reference. What exactly am I supposed to see?*

A: The following "Draw Mode Definitions" section shows the different drawing modes supported by XVT. There is also an explanation of what these modes will do if you are drawing in black or white on either a black or a white source pixel.

See Also: For more information, see "Draw_Mode" under "XVT Data Types" in the XVT Portability Toolkit Reference.

Note: On systems that use a 256-color palette, and not 24 bit color, information in the charts will hold true only for black and white because the palette indices are used for ORing and XORing, not the color values themselves. Because there are no definitive (or at least portable) rules about what color is held in a given index, there are absolutely no guarantees as to what your results will be. 129 xor 1 will always be 128, but index 129 might be yellow, 1 might be white, and 128 might be off-puce. The application can attempt to force a palette, but the colors present will be a random mix based on what applications are currently running and what applications have run in the past in the same session.

The following code and draw mode definitions demonstrate the problem more clearly:

```
typedef enum {   /* drawing (transfer) modes */
        M_COPY,
        M_OR,
        M_XOR,
        M_CLEAR,
        M_NOT_COPY,
        M_NOT_OR,
        M_NOT_XOR,
        M_NOT_CLEAR
} DRAW_MODE;
```

Draw Mode Definitions

M_COPY:

- If you draw black, source pixel will be forced to black
- If you draw white, source pixel will be forced to white

M_OR:

- If you draw black, source pixel will be forced to black
- If you draw white, source pixel will be left as is

M_XOR:

- If you draw black, source pixel will be inverted
- If you draw white, source pixel will be left as is

M_CLEAR:

- If you draw black, source pixel will be forced to white
- If you draw white, source pixel will be left as is

M_NOT_OR:

- If you draw black, source pixel will be left as is
- If you draw white, source pixel will be forced to black.

M_NOT_CLEAR:

- If you draw black, source pixel will be left as is
- If you draw white, source pixel will be forced to white

M_NOT_COPY:

- If you draw black, source pixel will be forced to white
- If you draw white, source pixel will be forced to black

M_NOT_XOR:

- If you draw black, source pixel will be left as is
- If you draw white, source pixel will be inverted

Q: *What is the difference between NText and CText?*

A: NText and CText each display a single line of text and provide alignment options within their frames. Although their basic functions are similar, each class has unique characteristics that make it better than the other in different situations.

The NText class is derived from the CNativeView class. Native views have the look-and-feel of objects provided by the native window manager. They look slightly different from platform to platform. Visually and functionally they fit in with the analogous graphical items on the target platform. They are not implemented by XVT-DSC++, but by native toolkits, so you have less flexibility in manipulating them. Native views don't know how to print themselves. Since native views are derived from CView, they have all of the capabilities of other objects at the view level. As a native view, NText is defined by platform-specific resources. For example, it uses the system font and color as defined by the window manager.

You should use NText when you want your application, or parts of your application (certain dialog boxes, for example), to have the look-and-feel of objects created by the native window manager.

The CText class is derived from the CView class. Unlike NText, which is drawn by the native window manager, CText creates drawn text which looks the same across all platforms. It allows user and program control over its font properties and colors. For example, it allows you to choose from a variety of font families (Times, Helvetica) and styles (italics, boldface). It can dynamically change its size as its contents change. It can change its placement and alignment at runtime. It can also output itself to a printer.

You should use CText when you want more creative control over the appearance of your text, when you want your text to appear the same across all platforms, or when you want to give the user creative control over the appearance of text in your application.

See Also: For more information, see "CText" and "NText" in the *XVT Development Solution for C++ Reference* and also look for references to CText and NText in the *Guide to XVT Development Solution for C++*.

The "Textual Views" chapter in *Introduction to C++ for Developers* is also helpful.

Q: *Is there a way to implement zooming in DSC++?*

A: The following solution does not use CUnits and will result in correctly updated wireframes, scrolling, sizing, dragging, and so on.

Create a new class called ViewInfo, for example. The purpose of ViewInfo is to keep track of the location where the view was created. Each time that a new view is inserted in the CScroller, create an associated ViewInfo. Fill the associated ViewInfo with the view's creation-frame and a pointer to this view. This ViewInfo instance is then appended to a RWOrdered.

When the zoom factor changes, for example, to 150%, iterate through the RWOrdered, and tell the view, which is pointed to size 1.5 times its original frame. Once all views have processed, call xvt_dwin_invalidate_rect on the CScroller. Everything should successfully redraw. If a CWireFrame has been moved, it generates a WFSizeCmd, and the DoCommand looks up in the RWOrdered to update the creation coordinates according to the actual zooming factor.

The following code illustrates:

```
class ViewInfo : public RWCollectable
{
public:
        ViewInfo( CView* theView, const CRect& theRect ) ;
        ~ViewInfo( ) ;
        virtual CRect& SetFrame( const CRect& theRect ) ;
        virtual CRect GetFrame( void ) ;
        virtual CView* GetView( void ) ;
protected:
        CRect itsCreationFrame;
        CView* itsView;
private:
}  ;
```

A fundamental problem is equating the Size() method with zooming. Here are the issues:

- What happens when a view is resized in the usual way? For example, as a pane in a splitter window, a subview may be resized to be twice as wide. Is this equivalent to zooming by 200%?
- What happens when a view is moved in the usual way? Will the associated ViewInfo object need to refresh itsCreationFrame? How would this be done?
- What happens when a native control is zoomed? For example, if a NListButton is told to zoom (resize), the edit box will remain the same height.
- What happens when a CPicture (or a CPictureButton, etc.) is told to resize? Will the picture stay intact?
- What happens to subviews within subviews? The splitter will be resized, but the oval will stay the same.

It should be possible to resolve all of these issues without the need to subclass everything. Expand on what has been started in the ViewInfo class above, and envision a type of visitor attached to the switchboard called a CZoomHandler.

A CZoomHandler will have a zooming factor attribute. If this is set to 100%, it will not do anything. A CZoomHandler will intercept E_UPDATE events at the Switchboard and perform a deep traversal through the window's object hierarchy, via DoDraw(). The CZoomHandler will render each view as it sees fit: On some views, it may just temporarily reset its size attributes and then call its Draw() method. On others, it may do its own drawing to handle some of the tougher issues listed above.

Q: *How do you create global variables for use in a DSC++ application?*

A: The best way to use variables that can be accessed globally from your application is to use them in a real global object, such as the CApplication- derived object. The application object should encapsulate the variables and make them accessible only through member functions. For instance, if the application object has a private variable named theVariable, then the application object might have a member function named SetTheVariable() and another called GetTheVariable(). This approach is a standard mode of operation in most object-oriented applications.

Some prefer to use the CGlobalUser class. This class, however, does not encapsulate and protect data as well as using a more object-oriented approach as described above. In case you choose to use the CGlobalUser class, the following paragraphs describe how.

The CGlobalUser class object has application global scope and can be used to access any global variables you may need. You can find documentation for this class in the *XVT Development Solution for C++ Reference*.

The CGlobalUser class is utilized as follows:

1. Copy the file **CGlobalUser.h** from the **pwr/include** directory to your development directory. You should rename the original file so that the compiler will see your own copy.
2. Add public class variables to your copy of the header file as follows:

```
//////////////////////
//Add items as needed//
//////////////////////

class CGlobalUser : public CNotifier
{
      public:
            CGlobalUser(void) {}
            XVT_HELP_INFO xd_help_info;
            FILE_SPEC* initFile;
            SECURITY_LEVEL userLevel;
};
```

3. In your application's startup member function, create an instance of CGlobalUser and pass it to CBoss: IBoss as follows:

```
////////////////////////////////
// Call IBoss to instantiate //
// the CGlobalUser object    //
////////////////////////////////

void CDEMOApp::StartUp();
{
        CApplication::StartUp();
        IBoss(new CGlobalUser);
        DoNew();
}
```

4. Access the global variables through the CBoss's GU pointer, as follows:

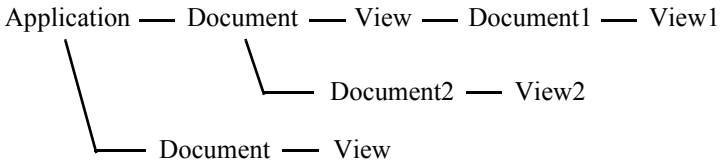
```
...
// Access the global userLevel
GU->userLevel = SUPER_USER;
...
```

5. Destroy the GU pointer in the application's ShutDown member function, as follows:

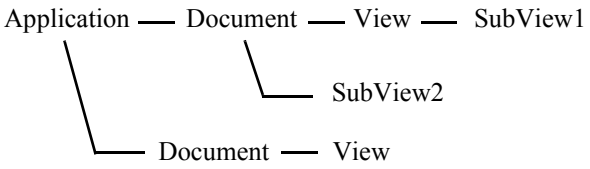
```
////////////////////////////////////
// Destroy GU and set it to NULL  //
////////////////////////////////////

void CDEMOApp::ShutDown(void)
{
        delete GU;
        GU = NULL;
        CApplication::ShutDown();
}
```

Q: *In the Application-Document-View hierarchy, can I have more levels of Document-View? For example, can I have a hierarchy like the following:*



In other words, if there are only three levels in the hierarchy, I have to put all data access/management code in one document and then use this single document to maintain all its views, as illustrated below?



A: No, you cannot have multiple levels of documents using the DSC++ framework. CDocument objects must be parented to one CApplicaton instance, just as CWindow objects are parented to a single CDocument instance. However, this arrangement gives you plenty of power for managing document data.

It might help to make a distinction between two different concepts that are used in the DSC++ framework. One is the "Application-Document-View" concept, and another is known as the "Model-View-Controller" design pattern. These two patterns can be used separately or together to build your application's data-flow structure.

It is true that you have only one level of views that are windows into the data in a document. However, it makes sense that there is only one level of complexity in this model. The real purpose of the App-Doc-View idea is to help the developer visualize which windows are looking at which separate groups of data.

In the App-Doc-View paradigm, it is the document's role to be the conduit of data flow between the data level and the presentation level of a two- or three-tier architecture. A document represents, in all its complexity, an entire, independent data set. Even if your presentation draws its data from several different sources, it can still be thought of as one data set, managed by a single document.

The App-Doc-View concept helps in laying out applications that have many windows that look into one data set, and a separate collection of windows that look into an entirely different data set. In your case, you may not have this type of complexity. More complex documents probably should be broken up into more manageable models, where the document manages (creates and destroys) these models. Each model is designed to solve one piece of the overall project.

In some cases, you may have a single window that looks into two separate data sets. In such a situation, the "Model-View-Controller" design pattern will be more appropriate. This design is borrowed

from the Smalltalk programming environment to help keep all the windows into a data set in sync so they all have the same data at the same time.

An MVC object structure can be as complicated as needed. When the state of one model changes, all other dependent models may be automatically notified and updated via the controllers to which the models are registered.

You can implement this with a document that owns many data models (use the CModel class). Each model has a controller that decides whether windows can change or read the data. You register each of the views with the data controller (use CController). These views can be implemented as CViews, CSubviews, or CWindows. When the data in the model changes, the controller will send a message to the appropriate views so that they can update themselves with the data. The document would manage both the data models and the views themselves.

In Architect, you can visualize the layout with the Application-Document-View graph. However there is no visual way to represent the Model-View-Controller idea in Architect because this design pattern has less to do with the layout of the application, and more to do with the internal data structures.

These two separate concepts have their own unique uses as generic design patterns. Thinking about object-oriented programs in terms of abstract design patterns has proven quite useful to many object-oriented programmers. A good book on the topic is *Design Patterns: Elements of Reusable Object-Oriented Software* by Gamma, Helm, Johnson, and Vlissides.

Q: *How do you print hidden views, multiple pages, or native controls in DSC++?*

A: The default behavior of Architect-generated code is to print the contents of a window on a single print page. DSC++ has built-in functionality for printing the screen images of most drawn or rendered objects, including CSubView-derived objects. Anything which lies beyond the boundaries of the window is clipped in the printed output. There is no functionality in DSC++ or the XVT Portability Toolkit (PTK) for printing images of native controls, specifically anything that inherits from the class CNativeView. If your application only needs to be able to produce simple screen shots of custom views drawn on a single page, you probably do not need to override any printing methods.

However, many applications need to be able to print text or graphics on multiple pages. Others may need to print portions of the view that demonstrate how to print the contents of a text editor object on multiple pages.

To understand Printing in DSC++, consider that there may be several overloaded versions of DoPrint acting in a single print process. CPrintManager has the DoPrint member function, CDocument has DoPrint, and CView also has DoPrint. These three implementations coexist, and they do different tasks. If you look at the DSC++ hierarchy, you can see they cannot override one another.

Printing is started when the user selects the Print option from the File menu. This generates the standard M_FILE_PRINT menu command. The menu command goes to your window's DoMenuCommand method. It propagates up from there to the default CWindow menu, then the default CDocument menu command. CDocument will then call the CDocument-derived DoPrint Method.

If your application overrides the CView::DoPrint, note that your overridden DoPrint function will not get called with the default Architect code. This is because the CView class does not inherit from CDocument, and it is the CDocument DoPrint that gets called by default. You will need to add your own code to call the proper print method. Usually this code is added at the window object level.

At the document level, the DoPrint method inserts each of the document's windows as an entry (or page) in the print queue, and calls the CPrintManager::DoPrint. The CPrintManager's DoPrint starts a PTK print thread. If you override CView::DoPrint, your function will also call CPrintManager::DoPrint.

The PrintThread function looks at every item in the print queue and opens a print page for each one. It calls an item's DoPrintDraw and then closes the page. This way each view in the queue gets exactly one page.

The default DoPrintDraw, generally at the CView level, simply sets the output device to be the printer, prepares the clipping region of the view, and calls the view's PrintDraw. PrintDraw is not called if the view is invisible. PrintDraw then does the drawing to the print page. In many cases, PrintDraw just calls Draw, the same routine that draws to the screen. Drawing to the screen or to the print page works interchangeably, depending on how the output device is set.

The secret to printing multiple pages it to override the DoPrint method that inserts pages into the print queue. Every time the CPrintManager's Insert method is called results in a page of printed output. If you want

a view to appear on several pages, call Insert once for each page with the same view as its parameter. If the objects to be printed are within a virtual frame, you can scroll hidden views into the visible portion of the frame before you enter the views in the print queue.

The overridden DoPrint also needs to figure out how many pages a view will occupy. Often times, this requires converting from printer dot units to pixel units. For this, you need to create a units object with dynamic mapping for your application.

Q: *How can I set an accelerator on the Mac for a menu item?*

A: The MENU_ITEM data structure provides a non-portable field for the Macintosh, which will allow accelerators to be set for single menu items in the menubar, as well as for single menu items in one of the pull-down menus.

Each MENU_ITEM data structure represents each menu item in the menubar or each menu item in a pull-down menu, as follows:

```
typedef struct s_mitem {
  MENU_TAG tag;
  char *text;
  short mkey;
  short accel;
  unsigned enabled:1;
  unsigned checked:1;
  unsigned separator:1;
  struct s_mitem *child;

  /* non-portable fields */

} MENU_ITEM;
```

accel is defined in **xvt_plat.h** through XVT_NP_MENU_FIELDS at the end of the MENU_ITEM structure in **xvt_type.h**

Additionally, in the Portability Toolkit, there are Mac specific fields in the MENU_ITEM structure for style and script.

Q: *The XVT Portability Toolkit for Macintosh requires my application to have the isHighLevelEventAware SIZE resource flag set. Does this mean that the XVT Portability Toolkit provides support for high-level Apple events?*

A: The XVT Portability Toolkit does not provide direct coupling of high-level Apple events to XVT events, but it assists in the implementation of high-level Apple events if you choose to support these yourself. This assistance comes in the form of calling

AEProcessAppleEvent from within the XVT Portability Toolkit's internal event handler. AEProcessAppleEvent, simply put, will intercept an incoming high-level Apple event and dispatch it to the appropriate registered handler.

By default, XVT installs (and subsequently removes) high-level Apple event handlers for kAEOpenDocuments and kAEPrintDocuments events. This is done within xvt_app_create so that any files to be opened or printed on application startup are processed and made available through the existing xvt_app_get_file mechanism. Therefore, if you need to provide your own handlers for kAEOpenDocuments and kAEPrintDocuments you should do it before xvt_app_create is called. This way XVT will not install its kAEOpenDocuments and kAEPrintDocuments handlers, and your application will have a single unified way of handling these events. The same is true for the kAEOpenApplication event and any other type of high-level Apple event.

XVT also installs a high-level event handler for kAEQuitApplication that is not removed. For this reason it is not possible for you to install a handler for kAEQuitApplication after xvt_app_create has been called. In addition, it is not advisable for you to install a handler for kAEQuitApplication before xvt_app_create is called since XVT needs to correctly terminate the application. If you supply a kAEQuitApplication handler and call exit() or ExitToShell() without correctly re-entering the application context, you will quit the Finder, not the application. Also, the XVT supplied kAEQuitApplication handler dispatches an E_QUIT event to the TASK Window event handler. Your application should perform quit application tasks in response to this event for portability rather than installing your own native high-level event handler. We hope that it is clear, now, why XVT strongly suggests that you do not use user-supplied kAEQuitApplication handlers.

The two main steps involved in implementing high-level Apple events are the creation of the high-level event handlers and the registration of those handlers to the high-level Apple events. The following code snippets demonstrate the creation of the high-level event handlers. These shells are for three core events: kAEOpenApplication, kAEOpenDocuments, and kAEPrintDocuments, but the concept can be carried forward for any high-level Apple event. (kAEQuitApplication is also core event, but we handle this for you.)

```
/* this code can be declared anywhere convenient */

#include <AppleEvents.h>
#include <GestaltEqu.h>

#define kGestaltMask 1L

pascal OSErr DoOpenApp (AppleEvent *theAppleEvent,
        AppleEvent *reply, long refCon)
{
   /* place your code here */
   return(/* handler needs to return correct
            OSErr result code */);
}

pascal OSErr DoOpenDoc (AppleEvent *theAppleEvent,
        AppleEvent *reply, long refCon)
{
   /* place your code here */
   return(/* handler needs to return correct
            OSErr result code */);
}

pascal OSErr DoPrintDoc (AppleEvent *theAppleEvent,
        AppleEvent *reply, long refCon)
{
   /* place your code here */
   return(/* handler needs to return correct
            OSErr result code */);
}
```

Once the high-level event handlers are declared, they must then be registered to the appropriate high-level Apple events. This typically is done before calling xvt_app_create. Be sure to make it the first call if possible so you don't miss any incoming high-level events. AEInit demonstrates the registration process.

```
void AEInit (void)
{
   OSErr err;
   long feature;

   err = Gestalt(gestaltAppleEventsAttr, &feature);
   if (err != noErr) {
      fprintf(stderr, "Problem in calling Gestalt!");
      return;
   }

   if (feature & (kGestaltMask << gestaltAppleEventsPresent)) {
      err = AEInstallEventHandler( kCoreEventClass, kAEOpenApplication,
(AEEventHandlerUPP)DoOpenApp, 0L, FALSE);
      if (err != noErr) fprintf(stderr, "AEOpenApplication Apple event not available!");

      err = AEInstallEventHandler( kCoreEventClass, kAEOpenDocuments,
(AEEventHandlerUPP)DoOpenDoc, 0L, FALSE);
      if ( err != noErr) fprintf(stderr, "AEOpenDocuments Apple event not available!");

      err = AEInstallEventHandler( kCoreEventClass, kAEPrintDocuments,
(AEEventHandlerUPP)DoPrintDoc, 0L, FALSE);
      if ( err != noErr) fprintf(stderr, "AEPrintDocuments Apple event not available!");
   }
   else {
      fprintf(stderr, "Apple events not available!");
   }
}
```

Note: This FAQ makes the assumption that you are familiar with implementing high-level Apple events natively and makes no attempt to be a definitive reference. For more information regarding high-level Apple events, please refer to “Inside Macintosh: Interapplication Communication.”

Q: *The printer resolution values returned by xvt_app_escape(XVT_ESC_GET_PRINTER_INFO, ...) are incorrect for some printers, such as the LaserPrinter, for which the values always return 72x72 dpi rather than 300x300 dpi. Initial research shows that some printers have more than one possible resolution and the Mac Toolbox seems to be returning the first one in the list, rather than the currently specified resolution.*

Is there a way to increase the printing resolution for printers that support it?

A: The following discussion is a response from Apple Developer Support in response to questions regarding printer resolutions:

The fundamental idea behind the Macintosh printing architecture is device independence combined with QuickDraw's built-in 72 dpi

resolution. The consequence is that an application "prints" by drawing into a standard 72 dpi grafport, whose portBits.bounds rectangle happens to represent the imageable area on the paper. Everything else, including the whole scaling business, is the job of the printer driver. (Note that the user may have chosen a scaling factor different from 100% in printer drivers that support scaled printing!) For example, the printer driver of the StyleWriter (360 dpi) by default requests for each font always 5 times the size specified by the application, in order to achieve the best possible text printing quality.

This 72-dpi principle also applies to QuickDraw pictures. As a matter of fact, it is conceptually nearly the same thing as recording drawing instructions in a PICT (between OpenPicture and ClosePicture calls), or as recording them in a printing port (between PrOpenPage and PrClosePage). Some applications actually prefer creating a picture for each page to be printed, and just call DrawPicture() into the printing port.

The limitation of this approach is that applications cannot take advantage of higher device resolutions. In order to give them a device independent way of imaging at one of the available device resolutions, the PrGeneral call has been introduced, with related opCodes, GetRsl, and SetRsl. If you use PrGeneral to set the resolution to something supported by the currently selected printer driver, the prInfo.iVRes and prInfo.iHRes values will reflect this setting, and the rPage rectangle corresponds to the new coordinate system in the printing port. Now, it's the responsibility of the application to scale everything up to the new coordinate system. On the other hand, the application can now control every pixel of the output device.

Regarding the (PostScript) LaserWriter, the context is more confusing because of PostScript, which is in itself a device-independent imaging model (with floating-point coordinates). The PS LaserWriter driver, in principle, only sees PostScript and lets PostScript figure out how to map graphic entities into the device pixels. This way, the LaserWriter driver can be used to print to many other PostScript devices of *any* resolution. Still, the 300dpi knowledge is built into it, and application developers may decide to set the resolution to this value, regardless of the features of the actual printing engine, just because they know it's the resolution of most of the (current) LaserWriters. Note that for some purposes, it may be more appropriate to set the resolution to 288 dpi (4*72), to avoid Moire effects.

In conclusion, if you don't have very specific requirements, just stay in the 72 dpi world, and let the printer drivers take care of everything. If you do have specific requirements (hair lines, your own halftoning etc.), use PrGeneral and scale everything according to the new coordinate system. See the article “Meet PrGeneral” in *Develop*, Apple's technical journal, issue 7F3, for a more detailed discussion and sample code.

In response to this information it has been decided that XVT cannot predetermine the printing needs of any application and use the device-independent printer calls. XVT does need to make sure the application can access this capability via non-portable system-specific calls, so to support this xvt_vobj_get_client_rect and xvt_vobj_get_outer_rect, when called on an XVT print window, will return the page rectangle and paper rectangle, respectively, in terms of the current printer's pixel resolution setting. (This is the rPage rectangle in the Mac TPrInfo record, and the rPaper rectangle in the Mac TPrint record.) The origin of the coordinate system is the top, left point of the page rectangle.

XVT has also tested the following code fragment as a way to set the printer resolution and keep this information consistent with the XVT print record. Any calls to xvt_app_escape(XVT_ESC_GET_PRINTER_INFO, ...) after updating the printer resolution, will reflect the correct information.

In most cases, the developer will want to implement this code fragment right after the XVT print record has been created. DSC++ users will want to consider overriding CApplication::GetPrintRecord and replacing calls to xvtPrintRcd with ((CApplication*)this)->itsPrintRecord.

The following code fragment makes the assumption that the application's XVT print record has been created, is valid, and is declared as PRINT_RCD *xvtPrintRcd:

```
/* Include the Mac headers before XVT headers */
#if (XVTWS == MACWS)
#include <Memory.h>
#include <Printing.h>
#endif

#if (XVTWS == MACWS)
  {
    THPrint printH;
    TSetRslBlk setRslBlk;

    /* Allocate a Mac relocatable print record */
    if ((printH = (THPrint)NewHandle(
                        sizeof(TPrint))) == NULL)
      xvt_dm_post_error("Out of memory");

    /* Copy existing data from XVT print record into
                        new print record */
    HLock((Handle)printH);
    memcpy(*printH, xvtPrintRcd, sizeof(TPrint));
    HUnlock((Handle)printH);

    /* Set up the data block to set the printer
                        resolution */
    setRslBlk.iOpCode = setRslOp;
    setRslBlk.iError = noErr;
    setRslBlk.hPrint = printH;
    setRslBlk.iXRsl = 300;
                        /* Set horizontal resolution */
    setRslBlk.iYRsl = 300;
                        /* Set vertical resolution */
    PrOpen();
    PrGeneral((Ptr)&setRslBlk);
    PrClose();
    if (setRslBlk.iError != noErr)
      xvt_dm_post_error("PrGeneral() error %d",
                              setRslBlk.iError);

    /* Update the XVT print rcd with any new data */
    HLock((Handle)printH);
    memcpy(xvtPrintRcd, *printH, sizeof(TPrint));
    HUnlock((Handle)printH);

    /* Dispose a Mac relocatable print record */
    DisposeHandle((Handle)printH);
  }
#endif
```

The above code assumes the current printer can handle 300 dpi resolution. This is not true for all printers, so there are two things your application can do. If you set the printer resolution to something it cannot handle, it will force the printer driver to use the default resolution. This is also a method of unsetting the resolution by specifying 0,0 as the resolution. You can also query the current printer to see what printer resolutions it can physically be set to using PrGeneral as follows:

```
#if (XVTWS == MACWS)
  {
   TGetRslBlk getRslBlk;
   getRslBlk.iOpCode = getRslDataOp;
   getRslBlk.iError = noErr;
   PrOpen(); PrGeneral((Ptr)&getRslBlk); PrClose();

   if (getRslBlk.iError != noErr)
     xvt_dm_post_error("PrGeneral() error %d",
                       getRslBlk.iError);

   /* Place your code here */
  }
#endif
```

The TGetRslBlk record contains the resolutions available on the current printing device. For information on how to use the TGetRslBlk record with the PrGeneral procedure, see "Determining and Setting the Resolution of the Current Printer" in *Inside Macintosh: Imaging With QuickDraw*.

Q: *How can I override the About box in DSC and DSC++?*

A: For XVT-DSC, there are two ways to override the About box. If you don't want to add to the controls in the box, but you would like to slightly modify the look, look in the **url_plat.h** file for the URL code that describes the About box. You can delete controls from this dialog, but you cannot add to them, and XVT does not recommend deleting controls from the About box.

The second, better way to override the About box is to create your own menu hook function that can be called from ATTR_MAC_MENU_HOOK. In the prototype for this function, use only the second and third parameters (macID and macItem). you can ignore the rest. XVT's internal representation for the Apple menu is macID = 32767. The first item on the menu, the About box item, is macItem = 1. Make sure your menu hook function returns FALSE if the macItem = 1 item is selected and returns TRUE for all others. You can test for these values in the menu hook function and call your own dialog or window creation function (i.e., xvt_dlg_create_res or xvt_win_create_res). Make sure that the container (your About box window) is modal. One approach is to make it a modal window. You will then have full access to the event handler to draw graphics, display QuickTime movies, etc.

For XVT-DSC++, the methods described above are always an option. However, the DSC++ framework provides a method that can be easily overridden, CApplication::DoAboutBox. This method is called in response to a user selection from the menu. You may override the

method to get your own About box by using an XVT-DSC++-type window instead of an XVT-PTK-type window. If possible, make the window modal. Finally, do not call the inherited object. Doing so will bring up the standard About box. For an example of overriding CApplication::DoAboutBox, download the DSC++ examples from XVT's FTP site.

XVT/MAC

INDEX

A

About box, B-27
Action Code Editor (ACE), 2-4
active window, A-2
Apple events, B-20
Apple, See Macintosh
application
    cursor control, A-12
    data and resource forks, 3-1
    timers, A-30
    title, 2-2
application programming
    extensibility, 2-2
    multibyte characters, 2-5
    optimizing performance, 3-4
    providing help for users, 3-10–??
    simulating color depth of 2, A-23
    system initialization, A-22
Application-Document-View concept, B-17
ATTR_ERRMSG_FILENAME, B-2
ATTR_ERRMSG_HANDLER, B-3
ATTR_EVENT_HOOK, A-25
ATTR_HAVE_COLOR, A-26
ATTR_HAVE_MOUSE, A-26
ATTR_KEY_HOOK, A-27
ATTR_MAC_ALWAYS_UPDATE, A-1
ATTR_MAC_BEHIND_WINDOW, A-2
ATTR_MAC_CHAR_TO_TASK, A-2
ATTR_MAC_CONTROL_HANDLE, A-3
ATTR_MAC_CTL_DEFER_UPDATE, A-3
ATTR_MAC_EVENT_TIME, A-4
ATTR_MAC_FOREIGN_WIN, A-4
ATTR_MAC_FRONT_WINDOW_FCN, A-4
ATTR_MAC_HAVE_COLOR_QUICKDRAW, 2-29, 2-31, A-5
ATTR_MAC_HILITE_MODE, A-5
ATTR_MAC_LBOX_KEY_HOOK, A-6
ATTR_MAC_LBOX_PROC_ID, A-7
ATTR_MAC_LOW_MEMORY_THRESHOLD, A-7
ATTR_MAC_MENU_HOOK, A-8, B-3
ATTR_MAC_MOUSE_CONTROL_FOCUS, A-9
ATTR_MAC_NATIVE_HTML_REFERENCE, A-9
ATTR_MAC_NO_GRAY_DISABLED_EDIT, A-10
ATTR_MAC_NO_GRAY_MAP_COLORS, A-10
ATTR_MAC_NO_LBOX_FOCUS_BOX, A-11
ATTR_MAC_NO_SELECT_WINDOW, A-11
ATTR_MAC_NO_SET_CURSOR, A-12
ATTR_MAC_NO_UPDATE_MENU_BAR, A-12
ATTR_MAC_PAT_RES_ID, A-13
ATTR_MAC_PAT_RES_INDEX, A-13
ATTR_MAC_PIXMAP_GWORLD_DEPTH, A-14
ATTR_MAC_PRINT_CLIPPING, A-14
ATTR_MAC_PRINT_COPIES, A-15
ATTR_MAC_PRINT_FIRST_PAGE, A-15
ATTR_MAC_PRINT_LAST_PAGE, A-16
ATTR_MAC_PROC_ID, 2-15, A-16
ATTR_MAC_ROUNDED_GROUPBOX, A-17

ATTR_MAC_SCROLL_THUMBTRACK, A-17
ATTR_MAC_SET_TITLE_AUTO_SELECT, A-18
ATTR_MAC_SHOW_JOB_DIALOG, A-18
ATTR_MAC_STR_HELP, A-19
ATTR_MAC_STR_STYLE_MENU1, A-20
ATTR_MAC_STR_STYLE_MENU2, A-20
ATTR_MAC_STR_STYLE_MENU3, A-21
ATTR_MAC_SYSTEM_INITIALIZATION, 2-4, A-22
ATTR_MAC_USE_COLOR_QUICKDRAW, A-23
ATTR_MAC_USE_NATIVE_ORIGIN, A-23
ATTR_MAC_WIN_MAX_HEIGHT, A-24
ATTR_MAC_WIN_MAX_WIDTH, A-24
ATTR_MAC_WIN_MIN_HEIGHT, A-24
ATTR_MAC_WIN_MIN_WIDTH, A-24
ATTR_MAC_WIN_USE_FIRST_CLICK, A-24
ATTR_MULTIBYTE_AWARE, A-27
ATTR_NATIVE_GRAPHIC_CONTEXT, A-29
ATTR_NATIVE_WINDOW, A-30
ATTR_NUM_TIMERS, A-30
ATTR_PRINTER_*, A-31
ATTR_R40_TXEDIT_BEHAVIOR, B-10
attributes
    non-portable, A-1
    portable, A-25
automatic dialog position flags, 2-16

B

Balloon Help, 2-26
bundle resources
    creator, 2-7
    data types, 2-7
    icons, 2-7

C

caret, where placed relative to selection, A-18
CGlobalUser.h file, B-15
CGrafPtr, A-29
character events, 2-5, A-2, A-6
cicn resource, 2-30, 2-31
clipping, A-14
CNTL resources, See controls, resources
color
    depth, A-14, A-23
    foreground and background, A-10
    highlight, A-5
    menus, 2-24
    monochrome, A-10, A-23
    system, A-26
    text, A-37
    using with controls, B-6
Color QuickDraw, A-5, A-23
color table
    menus, 2-25
    setting, A-37
colorized menus, 2-24
Command key, 2-26
compilation, conditional, 2-2, 2-23, 3-2
compile time optimization, B-9
compiler
    list of supported, 1-1
    Metrowerks CodeWarrior C++, 3-3
    optimization, 3-4
compiler options
    LANG_* CURL, B-8
Control key, 2-26
ControlHandle, A-3
controls
    creating, 2-14, 2-21
    defining constants, 2-20
    edit, A-10, A-18
    handle, A-3
    initialize or set as a group, A-3
    Macintosh icons, 2-29
    numbering, 2-19
    resources
        Default and Cancel buttons, 2-19
        dialog, 2-12
        min and max fields, 2-23
        procID field, 2-21
        refCon field, 2-23
        Rez definition, 2-20
        title, 2-23
        value field, 2-22
    rounded rectangle borders, A-17
    switching focus, A-9
    using color with, B-6
    using with dialog resources, 2-12

conventions
for code, 1-x
general manual, 1-ix
crsr resource, 2-29
CText, B-12
ctype.h file, 3-2
curl
automatic dialog positioning, 2-16
building in Metrowerks CodeWarrior, 3-6
menu accelerators, 2-26
menu creation, 2-24
native Macintosh string resources, B-4
curl.app
troubleshooting, B-1
curl.opt file, B-1
CURS resource, 2-29
cursor
how to control, A-12
resources, 2-29

D

data fork, 3-1
dctb resource, 2-17
DeRez, 2-30
development environment
Metrowerks CodeWarrior C++, 3-3
diacritical marks, A-6
dialog
automatic positioning, 2-16, A-31
color, 2-17, A-37
controls, adding color, 2-17
resources, 2-12
Rez definition, 2-13
setting color at runtime, 2-18
using controls, 2-12
DIALOG_POSITION constant, 2-16
directories, See folders
DITL resources, 2-12, 2-14
DLG_CANCEL, 2-20
DLG_OK, 2-20
DLOG resources, See dialog resources
doc folder, 1-1, A-1
document hierarchy, B-16
draw mode definitions, B-11
M_CLEAR, B-12
M_COPY, B-11
M_NOT_CLEAR, B-12
M_NOT_COPY, B-12
M_NOT_OR, B-12
M_NOT_XOR, B-12
M_OR, B-11
M_XOR, B-11
DRAW_CTOOLS, 2-36
DRAW_MODE, B-11

E

E_CHAR events, 2-5, A-2, A-6
E_MOUSE_DOWN events, A-9, A-11, A-24
E_UPDATE events, A-1, A-3
edit control
automatic selection of text, A-18
disabled appearance, A-10
native handle, A-33
error handling, B-4
error message handler
overriding, B-5
error messages, B-2
errscan
building in Metrowerks CodeWarrior, 3-6
escape codes, A-31
event
character, 2-5, A-2, A-6
hook, A-25
mouse down, A-9, A-11
native, A-25
thumbtrack, A-17
update, A-1, A-3
WaitNextEvent, A-4, A-25
Events.h file, A-25

F

fcntl.h file, 3-2
FILE_SPEC record, 2-4
files
CGlobalUser.h, B-15
creator, 2-7
ctype.h, 3-2
curl.opt, B-1
Events.h, A-25
fcntl.h, 3-2

icon.rsrc, 2-30
minit.c, 2-4, A-22
object, 3-2
QuickDraw.h, 2-19, A-29, A-30, A-37
readme, 1-1, A-1
required for XVT/Mac applications, 3-2
stdarg.h, 3-2
stddef.h, 3-2
stdio.h, 3-2
stdlib.h, 3-2
string.h, 3-2
time.h, 3-2
Types.h, 3-2
Types.r, 2-20
unix.h, 3-2
URL, 2-29
url.h, 2-9, B-1
url_plat.h, 2-10, B-3
Windows.h, 2-15
xvt.h, 3-2, B-1
xvt_env.h, 2-2
xvt_mctl.h, 2-20
finder icon resources, 2-31
floating window, A-4, A-11, A-24
focus
switching to different control, A-9
visual cue, A-11
folders
doc, 1-1, A-1
include, 2-2, 2-10
samples:ptk:mac, 2-4
, A-29
font
descriptor version identifier, 2-35
family resource name, A-32
logical, 2-35
native descriptors, 2-35
physical, 2-35
Font Style menu, A-20–A-22
fonts
mapping to multibyte fonts, B-8
standard, B-8
fork
data, 3-1
resource, 3-1
FrontWindow, A-4

G

geometry, PICTURE, A-36
global variables, B-15
graphics context, A-29
gray dithered patterns, A-10
group box control, appearance of, A-17
GWorld, A-14

H

help system, See online help
helpc
building in Metrowerks CodeWarrior, 3-6
helpview
building in Metrowerks CodeWarrior, 3-6
link libraries, 3-12
high-level Apple events, B-20
highlight color, A-5
hmnu Macintosh resources, 2-27
hypertext online help, See online help

I

ICON resource, 2-23, 2-30, 2-31
icon.rsrc file, 2-30
icons
as controls, 2-29, A-34
creating using xvt_dwin_draw_icon, 2-31
ictb resource, 2-17
IME, 2-5
include folder, 2-2, 2-10
initialization, 2-4
input method editor, See IME
internal warning messages
overriding, B-4
international characters, 2-5, A-6

J

Japanese characters, 2-5

K

key hook function, A-6
keyboard
accelerators, 2-26
international characters, 2-5
key translation, A-28
keys

Command, 2-26
Control, 2-26
Option, 2-26
Shift, 2-26

L

LANG_* CURL compiler options, B-8
LANG_GER_IS1, B-9
LANG_GER_W52, B-9
LANG_JPN_SJIS, B-9
language, See multibyte characters, Japanese characters
libraries
adding in Metrowerks CodeWarrior C++, 3-5
Mac Toolbox, 3-2
Metrowerks CodeWarrior C++, 3-5
required for XVT/Mac applications, 3-2
RogueWave Tools.h++, 3-5
text edit, 3-2, 3-5
XVTmPPCmwAPI.lib (core), 3-5
XVTmPPCmwHB.lib, 3-5, 3-11, 3-12
XVTmPPCmwHI.lib, 3-5
XVTmPPCmwPWR.libWR.lib, 3-5
XVTmPPCmwRW.lib, 3-5
XVTmPPCmwTE.lib, 3-5
XVTmPPCmwTES.lib, 3-5
XVT-Power++, 3-5
library, 3-11
link libraries, 3-5
list box
CNTL resource definition field, 2-21
definition ID, A-7
erratic behavior, B-4
focus control, A-11
font number set, A-33
keyboard key behavior, A-6
native handle, A-33
list edit
CNTL resource definition field, 2-21
native handle, A-33
logical fonts, 2-35
look-and-feel, 2-1, A-11, A-17, A-24, A-34
lowercase characters, 2-26

M

M_CLEAR, B-12
M_COPY, B-11
M_NOT_CLEAR, B-12
M_NOT_COPY, B-12
M_NOT_OR, B-12
M_NOT_XOR, B-12
M_OR, B-11
M_XOR, B-11
MAC_FLAG_*, 2-14, 2-22
mac_initialized, 2-4
MAC_STR_*, 2-34
MAC_STR_LOW_MEM_WARNING_ID, A-7
macID, B-3
Macintosh, 1-2
Balloon Help menu, 2-26
Color QuickDraw, A-5, A-23
cursor resources, 2-29
dialog automatic position flags, 2-16
Font Style menu, A-20–A-22
icons, 2-29
List Manager, B-4
look-and-feel, 2-1, A-11, A-17, A-24, A-34
menus, 2-23–2-25
PICT file format, A-35
QuickTime, B-3
resource compilers, 3-9
screen sizes and look-and-feel, 2-1
specific code, compiling conditionally, 2-3
Toolbox
accessing File Manager, 2-4
Dialog Manager, 2-20
initialization, 2-4
library, 3-2
WIND resource, 2-12
macItem, B-3
MacOS, 1-2
System 7, 2-9, 2-16
macros, when to invoke, 2-6
MACWS, 2-3
manual, conventions used in, 1-ix
mctb resource, 2-24
memory
approaching low threshold, 2-34, A-7
partition size, 2-9

menubars
compared to titlebars, 2-2
grayed (disabled), A-4
prevent updating, A-12
menus
"Quit" menu item, 2-24
accelerators, 2-26, B-20
Balloon Help, 2-26
color resources, 2-24
creating in Rez, 2-24
hierarchical, 2-24
hook function, A-8
M_HELP, 2-26
Macintosh, 2-23–2-25
non-portable colorized, 2-24
processing selection, A-8
updating, A-12
Metrowerks CodeWarrior C++
adding libraries, 3-5
building libraries, 3-6
development environment, 3-3
libraries, 3-5
optimization with XVT_OPT, 3-4
preferences, setting, 3-3
project files, 3-5
minit.c file, 2-4, A-22
modal dialogs
and online help, 3-11
ATTR_MAC_BEHIND_WINDOW, A-2
coding in Rez, 2-13
creating as Macintosh movable modal dialogs, 2-15
DLOG resource definition, 2-14
modal windows
ATTR_MAC_BEHIND_WINDOW, A-2
creating, A-34
modeless dialogs, 2-14
Model-View-Controller concept, B-17
monitors
color, A-26
monochrome, A-10, A-23
multiple, 2-1, A-32
Motif Window Manager, 1-2
mouse down event, A-9, A-11
mouse, checking for, A-26
MS-Windows 95, See Windows 95
MS-Windows 98, See Windows 98
MS-Windows NT, See Windows NT
multibyte characters, 2-5, 3-9, A-28, B-8
multibyte fonts, B-8

N

native
events, A-25
font descriptors, 2-35
functionality, 2-2
graphics device list, A-32
mouse down event, A-9, A-11
palette, A-35
pattern list resource, A-13
resources, 2-6
system initialization, A-22
system origin, A-23
window, A-30
NO_STD_BUNDLE, 2-6
NO_STD_HELP_MENU, 2-26
NO_STD_SIZE, 2-9
non-portable
attributes, A-1
code, 2-2
colorized menus, 2-24
escape codes, A-31
NText, B-12

O

online help
accessing, A-19
bound viewer
adding libraries in Metrowerks CodeWarrior C++, 3-5
building your application with, 3-10
help viewer
bound, 3-11
portable, 3-10
standalone, 3-12
libraries required, 3-2
M_HELP menus, 2-26
modal windows and dialogs, 3-11
standalone help viewer
adding libraries in Metrowerks Code-

Warrior C++, 3-5
optimization
Metrowerks CodeWarrior C++, 3-4
Option key, 2-26

P

pages, of printed output, A-15–A-16
palette, A-35
partition size, 2-9
PAT_SOLID, A-10
PAT_SPECIAL, A-13
patterns, A-10, A-13
performance, improving, 3-4
physical fonts
defined, 2-35
See Also font
PICT file format, A-35
PICT resource, 2-23
PICTURE
geometry, A-36
reading from file, A-35
picture control, A-34
pixmap, color depth, A-14
Portability Toolkit, See XVT Portability Toolkit
portable attributes, A-25
printing
clip region, A-14
current print record, A-31
dialog box, A-18
first page, A-15
hidden views, B-18
increasing resolution, B-23
last page, A-16
multiple pages, B-18
native controls, B-18
number of requested copies, A-15
project files, 3-5
PTK, See XVT Portability Toolkit

Q

QuickDraw.h file, 2-19, A-29, A-30, A-37
QuickTime, B-3

R

readme file, 1-1, A-1
rectangle
of desktop's current visible region, A-32
of window resized by user, A-24
ResEdit
adding color to dialogs, 2-17
adding color to menus, 2-24
creating dialogs, 2-12
creating icons, 2-29
cursor resources, 2-29
using to modify menus, 2-24
resource fork, 3-1
resources
and Universal Resource Language (URL), 2-6
BNDL (bundle), See bundle resources
cicn, 2-30, 2-31
CNTL (control), See controls, resources
crsr, 2-29
CURS, 2-29
dctb, 2-17
dialog, 2-12
DITL, 2-12, 2-14
errors finding, B-1
finder icon, 2-31
hmnu, 2-27
ICON, 2-23, 2-30, 2-31
ictb, 2-17
Macintosh
control, 2-19
cursor, 2-29
dialog, 2-12
strings, 2-33
mctb, 2-24
PICT, 2-23
SIZE, 2-9
'vers' (version), 2-11
window, 2-12
XVT/Mac, 2-6
Rez
using to code resources, 2-6
RogueWave Tools.h++ class library, 3-5

S

samples:ptk:mac folder, 2-4, A-29
SC_THUMBTRACK events, A-17
screen origin, A-23

scrollbar thumb position, A-17
Shift key, 2-26
SIZE resources, 2-9
source code, 3-6
SPCL:Main_Code tag, 2-4
SPCL:User_URL tag, 2-5
stack trace, B-2
stdarg.h file, 3-2
stddef.h file, 3-2
stdio.h file, 3-2
stdlib.h file, 3-2
string internationalization, 2-33
string lists, B-4
string.h file, 3-2
strings (Macintosh resources), accessing, 2-33
System 7, 2-9, 2-16

T

task window, A-2
text
    color, A-37
    highlighting, A-5
text edit
    CNTL resource definition field, 2-21
    dummy library, 3-5
    libraries, 3-2
    native handle, A-33
text edit objects, B-10
thumbtrack event, A-17
time.h file, 3-2
timers, A-4, A-30
titlebars, 2-2
toolbar, floating, A-24
transparent URL statement, 2-5
Types.h file, 3-2
Types.r file, 2-20

U

UNIX, 1-2
unix.h file, 3-2
update events, A-1, A-3
uppercase characters, 2-26
URL
    accel statement, 2-26
    coding
        dialog resources, 2-12
        menu resources, 2-23
        resources, 2-6
        SIZE resources, 2-9
        string resources, 2-33
    DIALOG statement and dialog position, 2-17
    file, 2-29
    FONT, FONT_MAP statements, 2-35
    transparent statements, 2-5
url.h file
    and SIZE resources, 2-9
    error involving, B-1
url_plat.h file, 2-10, B-3
userItem, 2-12, 2-20

V

'vers' (version) resources, 2-11

W

WaitNextEvent, A-4, A-25
warning messages
    overriding, B-4
WC_EDIT, A-33
WC_LBOX, A-33
WC_LISTEDIT, A-33
window
    active, A-2
    color, A-37
    creating new, A-2
    definition ID, A-16
    floating, A-4, A-11, A-24
    foreign, A-4
    geometry, A-24, A-32
    modal, A-34
    native, A-30
    print, A-14
    resources, 2-12
    setting color at runtime, 2-18
    task, A-2
WindowRef, A-30
Windows 95, 1-2
Windows 98, 1-2
Windows NT, 1-2
Windows.h file, 2-15

X

XVT Portability Toolkit, 1-1
    new features, B-8
xvt.h header file, 3-2, B-1
XVT/Mac, 1-2
    color mapping policy, A-10
    initialization, 2-4
    resource specifics, 2-6
    source code, 3-6
XVT/Win32
    supported platform, 1-2
XVT/XM, 1-2
xvt_app_create, 2-4, 2-34
XVT_COLOR_ACTION_SET, B-7
XVT_COLOR_ACTION_UNSET, B-7
XVT_COLOR_COMPONENT, B-7
xvt_ctl_create, A-11
xvt_ctl_create_def, 2-30, A-11
xvt_ctl_set_colors, B-6, B-7
xvt_dlg_create_res, 2-12
xvt_env.h file, 2-2
xvt_errmsg_sig, B-2
XVT_ESC_GET_PRINTER_INFO, A-31
XVT_ESC_MAC_DIALOG_POSITION, A-31
XVT_ESC_MAC_FONT_GET_RES_NAME, A-32
XVT_ESC_MAC_GET_DESKTOP_BOUNDS, 2-2, A-32
XVT_ESC_MAC_GET_DISPLAY_INFO, 2-2, A-32
XVT_ESC_MAC_GET_EDIT_HANDLE, A-33
XVT_ESC_MAC_GET_LIST_HANDLE, A-33
XVT_ESC_MAC_GET_PICT_ID, A-34
XVT_ESC_MAC_MODAL_WINDOW, A-34
XVT_ESC_MAC_PALET_GET_PALETTE_HANDLE, A-35
XVT_ESC_MAC_PICT_READ_FROM_FILE, A-35
XVT_ESC_MAC_PICTURE_COMMENT, A-35
XVT_ESC_MAC_RES_GET_PICT, A-36
XVT_ESC_MAC_SET_PICT_ID, A-36
XVT_ESC_MAC_SET_WINDOW_COLOR, 2-18, A-37
XVT_FILESYS_MAC, 2-3
xvt_font_set_native_desc, 2-35
xvt_mctl.h file, 2-20
xvt_menu_set_tree, A-12
xvt_menu_update, A-12
XVT_MINIMUM_MEM_SIZE, 2-9
XVT_OPT, B-9
    using in Metrowerks CodeWarrior C++, 3-4
XVT_PREFERRED_MEM_SIZE, 2-9
xvt_print_create_win, A-15, A-18
xvt_res_get_str, 2-33, B-4
xvt_res_get_str_list, 2-33, B-4
xvt_sbar_get_pos, A-17
xvt_sbar_set_pos, A-17
xvt_scr_set_focus_vobj(), B-10
xvt_timer_create, A-30
xvt_tx_set_active(), B-10
xvt_vobj_get_attr, A-1, A-31
xvt_vobj_move, A-3, A-24
xvt_vobj_set_attr, A-1, A-31
xvt_vobj_set_title, A-3, A-18
xvt_vobj_set_visible, 2-18
xvt_win_dispatch_event, A-25
xvt_win_set_ctl_colors, B-7
xvt_win_set_cursor, 2-29
XVT-Design
    Action Code Editor (ACE), 2-4
    coding resources with, 2-5
    Macintosh
        Toolbox initialization, 2-4
XVTmPPCmwAPI.lib library (core), 3-5
XVTmPPCmwHB.lib library, 3-5, 3-11, 3-12
XVTmPPCmwHI.lib library, 3-5
XVTmPPCmwPWR.lib library, 3-5
XVTmPPCmwRW.lib library, 3-5
XVTmPPCmwTE.lib library, 3-5
XVTmPPCmwTES.lib library, 3-5
XVT-Power++ class library, 3-5

Z

zooming in Power++, B-13