

OpenZinc

Addenda and Errata

Version 1

Addenda, Errata, and Technical Notes

**OpenZinc Application Framework
Version 1.0**

Copyright © 1990-1994 Zinc Software Incorporated
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU
Free Documentation License".

Table of Contents

1 Table of Contents vii

2 UIW_IMAGE 17

Class overview 18

Inheritance 18

Public 18

Types and constants 18

Member functions 18

Member variables 20

Protected 20

Member functions 20

Member variables 20

Class description 21

Public 21

Types and constants 21

Member functions 22

Member variables 25

Protected 25

Member functions 25

Member variables 25

Supported image formats 26

3 *ZIL_FILE* 27

Class overview 27

 Inheritance 27

 Public members 28

 Types, constants, and enums 28

 Member functions 28

 Protected 30

 Member variables 30

Class description 31

 Public members 31

 Types, constants, and enums 31

 Member functions 33

 Protected 36

 Member variables 36

4 *Addenda* 37

Changes to General notebook page of Combo box 37

Changes to General notebook page of Pop-up item 37

Changes to the Geometry page of OpenZinc Designer 38

 Combo box 38

 Constraints 39

 Anchor 39

New ability to access lists associated with objects 40

New help system 40

New inline functions for checking flags 41

 Using new functions instead of FlagSet() 41

 Using multiple functions instead of FlagsSet() 41

New mouse message 47

New open and print document events 48
 Implementing opening and printing functionality 48
 Restrictions on drag-and-drop under Macintosh 49
 Environment-specific mapping 49
 Using these events programmatically 49
 Using S_OPEN_DOCUMENT 50
 Using S_PRINT_DOCUMENT 51

New support for 3D controls in OpenZinc 1.0 51

Functions moved from Designer to the library 52

System events 53
 All system events available to the user 53
 Some logical events generic to all platforms 53

PowerPak 32 and crashing applications 54

5 *Errata* 55

Programmer's Reference Volume One—Window Objects 55

Introduction 55
 UIW_STRING inheritance 55

Chapter 1—UI_APPLICATION 55
 argc and argv 55

Chapter 7—UI_DISPLAY 56
 UI_DISPLAY::Text() 56
 Calling VirtualGet() and VirtualPut() 56

Chapter 12—UI_EVENT_MANAGER 56
 DeviceState() 56

Chapter 14—UI_GEOMETRY_MANAGER 56
 Pixel and minicell 56

Table of Contents

<i>Chapter 17—UI_HELP_SYSTEM</i>	57
Incorrect class name	57
<i>Chapter 20—UI_LIST</i>	57
Undocumented behavior of Add()	57
<i>Chapter 22—UI_MACINTOSH_DISPLAY</i>	57
maxColors	57
MapRGBColor()	57
patternTable and rgbColorMap	57
PTN_RGB_COLOR	57
FontRec **fRec	58
PTN_INTERLEAVE_FILL	58
<i>Chapter 32—UI_PRINTER:</i>	58
Dot-matrix printer support	58
Using UI_PRINTER	58
New BeginPage() function	60
New TextFormat() function	60
<i>CHAPTER 38—UI_RELATIVE_CONSTRAINT</i>	60
Centering flags	60
<i>CHAPTER 42—UI_WINDOW_MANAGER</i>	60
exitFunction	60
<i>Chapter 43—UI_WINDOW_OBJECT</i>	61
New member function and flag	61
<i>Chapter 53—ZIL_DECORATION_MANAGER</i>	61
New defaultOSName	61
<i>Chapter 57—ZILJ18N_MANAGER</i>	61
New defaultOSName	61
<i>Chapter 61—ZIL_LANGUAGE_MANAGER</i>	61
New defaultOSName	61
<i>Chapter 64—ZIL_LOCALE_MANAGER</i>	62
New defaultOSName	62

Programmer's Reference Volume Two—Window Objects 62

Introduction 62

Chapter 1—UIW_BIGNUM 62

Information requests 62

NMF_SCIENTIFIC not supported 62

Chapter 3—UIW_BUTTON 62

Toggling behavior 62

Toggling appearance 63

TOP is bottom edge 63

Chapter 5—UIW_DATE 63

Information requests 63

Chapter 14—UIW_POPJJP_ITEM 63

MNIF_ABOUT for Macintosh only 63

Chapter 19—UIW_REAL 63

NMF_DIGITS should be NMF_DECIMAL 63

Chapter 20—UIW_SCROLL_BAR 64

Default width and height 64

Chapter 22—UIW_STATUS_BAR 64

Adding objects to status bar 64

Chapter 24—UIW_SYSTEM_BUTTON 64

Using Apple menu "About" item 64

Chapter 25—UIW_TABLE 64

Building library with persistence 64

Chapter 29—UIW_TIME 64

Information requests 64

Chapter 33—UIW_WINDOW 65

New constructor 65

Setting WOAF_MDI_OBJECT 65

Using default storage with the storage constructor 65

Table of Contents

Chapter 35—ZAF_MESSAGE_WINDOW 66
New member variables 66

Getting Started 67

Chapter 4—Writing Multiplatform Programs 67
Different contents than listed 67
New libraries 67

Chapter 7—OpenZinc and C++ 67
Misspelling 67

Chapter 18—Using Languages 68
Code to switch windows wrong 68
CreateWin() function 68

Chapter 19—Program Design 68
Accelerator keys work differently 68

Appendix A—Compiler Considerations 68
New Mac_ZIL1 1 68

Appendix B—Example Programs 68
Description of VALIDT program 68

Appendix D—Keyboard and Mouse Mappings 69
Keyboard mappings for gray keys 69
Backspace and Delete 69
<Opt+Tab> 69

OpenZinc ***Designer 70***

Chapter 1—Introducing OpenZinc Designer 70
Reactivating object on Mac 70
Pull-down menu on Mac 70

Chapter 11—Edit Options 70
Grouping objects on Mac 70
Relative constraints in OpenZinc Designer 70

<i>Chapter 15—Control Objects</i>	71
Can't set Send flag for pull-down item	71
<i>Chapter 17—Other Objects</i>	71
Columns and RecordSize	71
Initializing table with DataSet()	71
Manipulating table visually	72
Window Object option in OpenZinc Designer	72
<i>Chapter 19—Image Editor</i>	73
Grouping objects	73
<i>Appendix A—Compiling the Designer</i>	73
Libraries	73
Defaults Editor	73
Integrated OpenZinc Designer modules	73
Drag and drop	73
<i>Mac and NEXTSTEP File and Edit menus</i>	76
<i>Microsoft and Watcom graphics</i>	77
<i>New virtualized Add() and Subtract() functions</i>	77
<i>Recommended stack size</i>	78
<i>Macintosh heap size</i>	78

6 OpenZinc Technical Notes 79

IDEs 80

Setting up the Microsoft IDE to work with OpenZinc	ZD-TN1006	83
Watcom 10.0 IDE for Windows	ZD-TN5006	85

Fonts 87

Changing object fonts	ZD-TN1001	88
Adding a new font to the fontTable under OS/2	ZD-TN1002	89
Adding a new font to the fontTable in GFX graphics	ZD-TN1003	90
Adding a new font to the fontTable in Motif	ZD-TN2001	92

Table of Contents

Adding a fixed-width font to the fontTable
under Windows ZD-TN5002 94

Event flow, messaging, and flags 95

Event flow ZD-TN1005 96
Local control loops ZD-TN1008 98
Interpretation of events in OpenZinc ZD-TN4572 100
Trapping the S_CLOSE event ZD-TN1010 103
Working with Rags ZD-TN7000 105

Drawing, video, and graphics 109

The DrawItem() function ZD-TN1007 110
Multithreaded applications on OS/2 ZD-TN1017 111
Changing video modes at run time ZD-TN2005 113
Palette mapping in OpenZinc ZD-TN4573 117
Loading large bitmaps with OS/2-specific functions ZD-TN1011 120
Setting the mouse cursor to a wait state ZD-TN4020 122

OpenZinc Designer 124

Integrating Directory Services from OpenZinc Designer ZD-TN1009 125
The Service Manager ZD-TN1015 128
Exporting a window to text ZD-TN1016 129
Using multiple .DAT files in one application ZD-TN2003 131
Security on a .DAT file ZD-TN2014 134
Loading a derived object from a .DAT file ZD-TN3010 136
Using a table created in OpenZinc Designer ZD-TN3011 137
Checking for errors when reading .DAT files ZD-TN2007 140
Deriving an object in OpenZinc Designer ZD-TN2009 141
Designer fails to launch ZD-TN2011 143
Designer import and export file types ZD-TN2012 144
Displaying information in a table created
in the Designer ZD-TN3011 145
Using the ZAF_STRING_EDITOR class
of the Designer ZD-TN4025 148

UI objects and programming techniques 151

Removing geometry constraints ZD-TN1014 152
Using a button to close a window ZD-TN2000 154
Changing object flag or status at run time ZD-TN2002 155
Check boxes or toggle buttons on a window ZD-TN2004 156

Nonselectable gray and nonselectable black	ZD-TN2013	157
Scrolling more than 32,767 records in a UIW_TABLE	ZD-TN2016	158
Columns in lists	ZD-TN3000	160
Retrieving the nth item from a list	ZD-TN3001	161
Adding an object at run time	ZD-TN3002	162
Moving an object at run time	ZD-TN3003	163
Removing the title from a window with the Designer	ZD-TN3004	164
Selecting a check box or radio button programmatically	ZD-TN3005	165
Lifetime of an object	ZD-TN3006	166
Forcing an invalid field to remain current	ZD-TN3007	168
Menus on the Macintosh	ZD-TN3015	170
Adding pop-up menus to the screen	ZD-TN3016	173
Upgrading applications to run on OpenZinc 1.0	ZD-TN3018	175
Deriving a thermometer-type object	ZD-TN4005	186
Using the dot matrix printer	ZD-TN4026	187
Differences in platforms—Windows	ZD-TN4031	189
Differences in platforms—OS/2	ZD-TN4033	190
Platform differences—Motif, Macintosh, NEXTSTEP, Curses	ZD-TN4034	192
MDI application guidelines	ZD-TN4035	193
DGROUP exceeding 64K	ZD-TN4036	194
Creating vertical lists with buttons, fonts, and centered text	ZD-TN5000	195
Inserting a new object into a vt. or hz. list at run time	ZD-TN5001	197
Drag and drop in 4.0	ZD-TN5003	198
Enabling table navigation using keyboard and mouse	ZD-TN5004	203
Working with UIW_TABLE_RECORD	ZD-TN2008	205
Displaying MDI children after adding to MDI parent	ZD-TN2010	208
Graying out a bitmap on a button when it is nonselectable	ZD-TN4016	209
Displaying geometric objects on a window	ZD-TN4017	210
Deriving a window	ZD-TN1013	212
Interfacing the UIW_TABLE object to a database	ZD-TN2006	213
Viewing system resources at run time	ZD-TN4018	215
User functions that are members of a class	ZD-TN4019	216

Table of Contents

Displaying timed window on application startup	ZD-TN4021	218
Using UIW_TABLE with a database	ZD-TN4026	219

OpenZinc and third-party libraries 222

MetaWINDOW display class	ZD-TN2015	223
General Protection Faults	ZD-TN3008	225
Eliminating optional library components	ZD-TN3009	227
Using PharLap 286 with OpenZinc	ZD-TN4004	228

Unicode and il8n 229

Decreasing the shipping size of Unicode.dat	ZD-TN4003	230
---	-----------	-----

Help and error system 236

Integrating native Windows help with OpenZinc	ZD-TN3013	237
---	-----------	-----

Index ccxli

UIW_IMAGE

UIW_IMAGE allows you to load a bitmapped image in native format from an image file, such as .tif or .pcx, or a resource file into your OpenZinc application. You can display these images in a window normally, tiled, or stretched—either in their own region of the window, or as nonfield regions occupying all available space within a window.

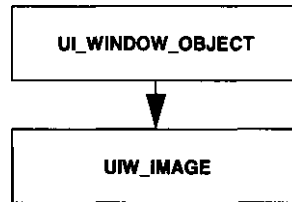
In environments that use resource files, such as DOS, Windows, OS/2, and Macintosh, load the image from the application's resource file, a native image file, or from the application's .DAT file. In environments that do not use resource files, such as OSF/Motif and NEXTSTEP, load an image from a native image file. Currently, UIW_IMAGE only loads native images; you cannot use a UIW_IMAGE object to load an image other than the image types specified in Table 1, "Supported bitmap image formats of UIW_IMAGE," on page 26, for specific operating environments. (In other words, don't ask an image object running under Windows to load an OSF/Motif bitmap.)

A UIW_IMAGE object cannot store its image data; it can only store its own data, such as flags, dimension, and other attributes. To store images, use another method, such as a native image manipulation program.

Because UIW_IMAGE derives from UI_WINDOW_OBJECT, an image object inherits the window object class's persistence mechanisms. This means that you can use the normal mechanisms for loading and storing image object descriptions in .DAT files.

Class overview

Inheritance



Located in	UI_WIN.HPP
Implemented in	Z_IMAGE.CPP
Portability	All except Curses, DOS text

Public

Types and constants

const IMF_FLAGS IMF_NO_FLAGS	= 0x0000
const IMF_FLAGS IMF_TILED	= 0x0001
const IMF_FLAGS IMF_SCALED	= 0x0002
const IMF_FLAGS IMF_BACKGROUND	=0x0004
const IMF_FLAGS IMF_STATIC_HANDLE	=0x0008
const IMF_FLAGS IMF_AUTO_SIZE	=0x0010

Member functions

```

UIW_IMAGE(int left, int top, int width, int height,
          ZIL_ICHAR *pathName,
          IMF_FLAGS imFlags = IMF_NO_FLAGS,
          WOF_FLAGS woFlags = WOF_NO_FLAGS);

virtual ~UIW_IMAGE(void);

virtual EVENT_TYPE DrawItem(const UI_EVENT &event,
                           EVENT_TYPE ccode);

virtual EVENT_TYPE Event(const UI_EVENT &event);

int IsBackground(void) { return ((imFlags & IMF_BACKGROUND)

```

```

    ? TRUE: FALSE);}

int IsScaled(void) {return ((imFlags & IMF_SCALED)
    ? TRUE : FALSE);}

int IsTiled(void) {return ((imFlags & IMF_TILED) ? TRUE: FALSE);}

int HasStaticHandle(void) {return ((imFlags & IMF_STATIC_HANDLE)
    ? TRUE: FALSE);}

int IsAutoSize( void) { return ((imFlags & IMF_AUTO_SIZE)
    ? TRUE: FALSE);}

#ifdef ZIL_LOAD

virtual ZIL_NEW_FUNCTION NewFunction(void) { return
    (UIW_IMAGE::New); }

static UI_WINDOW_OBJECT *New(const ZIL_ICHAR *name,
    ZIL_STORAGE_READ_ONLY *file =
        ZIL_NULLP(ZIL_STORAGE_READ_ONLY),
    ZIL_STORAGE_OBJECT_READ_ONLY *object =
        ZIL_NULLP(ZIL_STORAGE_OBJECT_READ_ONLY),
    UI_ITEM *_objectTable = ZIL_NULLP(UI_ITEM),
    UI_ITEM *_userTable = ZIL_NULLP(UI_ITEM))
{
    return (new UIW_BUTTON(name, file, object,
        _objectTable, _userTable));
}

UIW_IMAGE(const ZIL_ICHAR *name,
    ZIL_STORAGE_READ_ONLY *file,
    ZIL_STORAGE_OBJECT_READ_ONLY *object,
    UI_ITEM *objectTable = ZIL_NULLP(UI_ITEM),
    UI_ITEM *userTable = ZIL_NULLP(UI_ITEM));

virtual void Load(const ZIL_ICHAR *name,
    ZIL_STORAGE_READ_ONLY *file,
    ZIL_STORAGE_OBJECT_READ_ONLY *object,
    UI_ITEM *objectTable, UI_ITEM *userTable);

#endif

```


UIW_IMAGE

```
#if defined(ZIL_STORE)  
  
virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,  
                  ZIL_STORAGE_OBJECT *object,  
                  UI_ITEM *objecTable, UI_ITEM *userTable);  
  
#endif
```

Member variables

```
static ZIL_ICHAR _classNameQ;  
IMF_FLAGS imFlags;
```

Protected

Member functions

```
int LoadImageFromApplication(void);
```

```
int LoadImageFromFile(void);
```

Member variables

```
ZIL_ICHAR *pathName;
```

```
ZIL_IMAGE_HANDLE image;
```

```
int imageWidth, imageHeight;
```

Class description

Public

Types and constants

IMF_NO_FLAGS

Causes a new or existing UIW_IMAGE object to exhibit default behavior, which is neither tiled nor scaled.

IMF_TILED

Instantiates a UIW_IMAGE object that, starting from the upper left-hand corner of the region, displays copies of itself to cover all of the region's available space. If the width and height of the object are such that multiple copies of the images cannot fit perfectly in the region, the tiled images will appear to display themselves beyond the right and bottom edges of the region area, leaving only portions of the image visible at the region's boundary.

IMF_SCALED

Instantiates a UIW_IMAGE object that occupies all available space in its region, whether or not the width and height of the region are bigger or smaller than the width and height of the image.

If the region width is smaller than the image width, the image will appear condensed horizontally; and if the region width is larger than the image width, the image will appear stretched out of proportion horizontally.

If the region height is smaller than the image height, the image will appear condensed vertically; and if the region height is larger than the image height, the image will appear stretched out of proportion vertically.

IMF_BACKGROUND

The UIW_IMAGE equivalent of a nonfield region. Instantiates a UIW_IMAGE object that occupies the size of its parent window as either tiled or scaled, depending on the flags passed in at construction time. Additionally, you can combine the IMF_BACKGROUND flag with the IMF_TILED or IMF_SCALED flags for a tiled or scaled background image. Cannot be used with IMF_AUTO_SIZE.

IMF_STATIC_HANDLE

Maintains the file handle to a UIW_IMAGE object, even if the object is destroyed. Useful for sharing a single image between multiple programs. If you construct an image object and load in an image file that already exists in memory and is used by

UIW_IMAGE

one or more other applications, the UIW_IMAGE object will set this file automatically.

IMF_AUTO_SIZE

Causes the parent window of a UIW_IMAGE object to size itself automatically to the size of the image object's region. Cannot be used with IMF_BACKGROUND.

Member functions

```
UIW_IMAGE(int left, int top, int width, int height,  
          ZIL_ICHAR *pathName,  
          IMF_FLAGS imFlags = IMF_NO_FLAGS,  
          WOF_FLAGS woFlags = WOF_NO_FLAGS);
```

Constructs a UIW_IMAGE object. Finds image data in the file called pathName. Default construction is to construct it in a window with the same proportions as the image. Pass in

- IMF_TILED
- IMF_SCALED,
- IMF_BACKGROUND, in conjunction with IMF_TILED or IMF_SCALED, to create a tiled or scaled image in a window, or
- IMF_AUTO_SIZE.

To use the member functions LoadImageFromApplication() and LoadImageFromFile(), specify a pathname for the image on disk. Use this constructor except when creating your own persistent UIW_IMAGE object.

```
virtual ~UIW_IMAGE(void);
```

Destroys a UIW_IMAGE object and frees any memory used.

```
virtual EVENT_TYPE DrawItem(const UI_EVENT &event,  
                            EVENT_TYPE ccode);
```

When this function is called, the UIW_IMAGE object virtualizes its screen region, displays itself, then devirtualizes its region. If the image has a border, it will draw itself with a border one pixel wide. Default construction is no border.

```
virtual EVENT_TYPE Event(const UI_EVENT &event);
```

Event() function of the UIW_IMAGE object. By default, the UIW_IMAGE object does not respond to any events.

```
int IsBackground(void) { return ((imFlags & IMF_BACKGROUND)
? TRUE: FALSE);}
```

Returns TRUE if the UIW_IMAGE was constructed as a background image. Returns FALSE otherwise.

```
int IsScaled(vokl) { return ((imFlags & IMF_SCALED)
? TRUE: FALSE);}
```

Returns TRUE if the UIW_IMAGE was constructed as a scaled image. Returns FALSE otherwise.

```
int IsTiled(void) { return ((imFlags & IMF_TILED) ? TRUE: FALSE);}
```

Returns TRUE if the UIW_IMAGE was constructed as a tiled image. Returns FALSE otherwise.

```
int HasStaticHandle(void) {return ((imFlags & IMF_STATIC_HANDLE)
? TRUE: FALSE);}
```

Returns TRUE if the UIW_IMAGE object was constructed with a static handle. Returns FALSE otherwise.

```
int AutoSizes() { return ((imFlags & IMF_AUTO_SIZE) ? TRUE: FALSE);}
```

Returns TRUE if the UIW_IMAGE was constructed to autosize. Returns FALSE otherwise.

```
#if defined(ZIL_LOAD)
virtual ZIL_NEW_FUNCTION NewFunction(void)
{
    return (UIW_IMAGE::New);}
#endif
```

Returns a new UIW_IMAGE object (ZIL_LOAD only).

```
static UI_WINDOW_OBJECT *New(const ZIL_ICHAR *name,
ZIL_STORAGE_READ_ONLY *file =
ZIL_NULLP(ZIL_STORAGE_READ_ONLY),
ZIL_STORAGE_OBJECT_READ_ONLY *object =
ZIL_NULLP(ZIL_STORAGE_OBJECT_READ_ONLY),
UI_ITEM *_objectTable = ZIL_NULLP(UI_ITEM),
UI_ITEM *_userTable = ZIL_NULLP(UI_ITEM))
{
    return (new UIW_BUTTON(name, file, object,
_objectTable, _userTable));
}
```

Returns a new read-only, persistent UIW_IMAGE object using the following parameters (ZIL_LOAD only).

UIW_IMAGE

name	name of the window object stored in the .DAT file
file	pointer to ZIL_STORAGE_READ_ONLY
object	pointer to ZIL_STORAGE_OBJECT_READ_ONLY
objectTable	table of pointers to New() functions of persistent objects
userTable	table of pointers to user functions of persistent objects

```
UIW_IMAGE(const ZIL_ICHAR "name,  
           ZIL_STORAGE_READ_ONLY *file,  
           ZIL_STORAGE_OBJECT_READ_ONLY *object,  
           UI_ITEM *objectTable = ZIL_NULLP(UI_ITEM),  
           UI_ITEM *userTable = ZIL_NULLP(UI_ITEM));
```

Returns a new read-only, persistent UIW_IMAGE object using the following parameters (ZIL_LOAD only).

name	name of the window object stored in the .DAT file
file	pointer to ZIL_STORAGE_READ_ONLY
object	pointer to ZIL_STORAGE_OBJECT_READ_ONLY
objectTable	table of pointers to New() functions of persistent objects
userTable	table of pointers to user functions of persistent objects

```
virtual void Load(const ZIL_ICHAR *name,  
                 ZIL_STORAGE_READ_ONLY *file,  
                 ZIL_STORAGE_OBJECT_READ_ONLY *object,  
                 UI_ITEM *objectTable, UI_ITEM *userTable);
```

Loads a read-only UIW_IMAGE from storage using the following parameters (ZIL_LOAD only).

name	name of the window object stored in the .DAT file
file	pointer to ZIL_STORAGE_READ_ONLY
object	pointer to ZIL_STORAGE_OBJECT_READ_ONLY
objectTable	table of pointers to New() functions of persistent objects
userTable	table of pointers to user functions of persistent objects

```
virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,  
                  ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,  
                  UI_ITEM *userTable);
```

Stores a UIW_IMAGE using the following parameters (ZIL_STORE only).

name	name of the window object stored in the .DAT file
file	pointer to ZIL_STORAGE_READ_ONLY
object	pointer to ZIL_STORAGE_OBJECT_READ_ONLY
objectTable	table of pointers to New() functions of persistent objects
userTable	table of pointers to user functions of persistent objects

Member variables

static ZIL_ICHAR _className[];

The class name of the current object.

IMF_FLAGS imFlags;

The attributes of the current object.

Protected***Member functions***

int DestroyImageHandle(void);

Destroys an image and deallocates any memory it may have used,

int LoadImageFromApplication(void);

Loads the image in native format from a resource file in an application,

int LoadImageFromFile(void);

Loads the image in native format from a file stored on disk.

Member variables

ZIL_ICHAR *pathName;

Path name of the read-only image on disk.

ZIL_BITMAPHANDLE image;

A handle to the image. Set by the operating environment

int imageWidth, imageHeight;

Width and height of the current image in pixels.

UIW_IMAGE

Supported image formats

Using UIW_IMAGE, you can display the following bitmap formats under these operating environments:

TABLE 1. Supported bitmap image formats of UIW_IMAGE

<i>Platform</i>	<i>Image format</i>
DOS graphics	.PCX, .DCX
Windows	.BMP or resource file
OS/2	.BMP or resource file
Macintosh	PICT
Motif	XBM or .XPM
NEXTSTEP	.TIFF

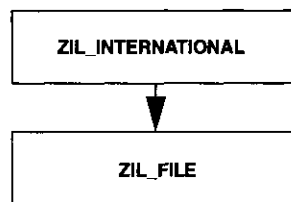
ZIL_FILE

In release 1.0, OpenZinc moved the functionality of the Designer class IO_FILE to the new library class ZIL_FILE, a general-purpose class for reading and writing files, including but not limited to .DAT files. ZIL_FILE writes and reads files to and from disk under all operating environments and locales that OpenZinc supports.

A ZIL_FILE object can write data in either text or binary format. Further, a ZIL_FILE object supports read-only or read-write attributes. It also reports any errors that occur when it saves or reads a file.

Class overview

Inheritance



Located in	UI_GEN.HPP
Implemented in	Z_FILE.CPP
Portability	All

Public members

Types, constants, and enums

```

typedef UIF_FLAGS UIS_FLAGS;
const UIS_FLAGS UIS_READ           = 0x0001;
const UIS_FLAGS UIS_READWRITE     = 0x0002
const UIS_FLAGS UIS_CREATE        = 0x0004
const UIS_FLAGS      UIS_OPENCREATE = 0x0008
const UIS_FLAGS UIS_TEMPORARY     = 0x0010
const UIS_FLAGS      UIS_COPY      = 0x0020
const UIS_FLAGS UIS_BINARY        = 0x0100
const UIS_FLAGS UIS_TEXT          = 0x0200

```

enum Z_ERROR

```

{
    ERROR_NONE    = 0,
    ERROR_NAME    = 1,
    ERROR_NULL_STRING = 2,
    ERROR_ACCESS  = 3
};

```

enum SEEK

```

{
    SEEK_FROM_START = 0,
    SEEK_FROM_CURRENT = 1,
    SEEK_FROM_END = 2
}

```

Member functions

```

ZIL_FILE(const ZIL_ICHAR *pathName,
          UIS_FLAGS access = UIS_READ | UIS_BINARY);

virtual ~ZIL_FILE(void);

long Length(void) const;

long Tell(void) const;

ZIL_FILE::Z_ERROR GetError(void);

```

```
ZIL_FILE::Z_ERROR SetError(ZIL_FILE::Z_ERROR error);

virtual ZIL_FILE::Z_ERROR Open(void);

virtual ZIL_FILE::Z_ERROR Close(void);

Z_ERROR Unlink(void);

Z_ERROR Rename(const ZIL_ICHAR *pathName);

Z_ERROR Seek(long offset, SEEK location) const;

virtual int Read(ZIL_INT8 *value) const;

virtual int Read(ZIL_INT8 **value) const;

virtual int Read(ZIL_INT16 *value) const;

virtual int Read(ZIL_UINT16 *value) const;

virtual int Read(ZIL_INT32 *value) const;

virtual int Read(ZIL_INT32 **value) const;

virtual int Read(ZIL_ICHAR *text, int length) const;

virtual int Read(ZIL_ICHAR **text) const;

virtual int Read(void *buffer, int size, int length) const;

#if defined(ZIL_UNICODE)

virtual int Read(char *text, int length) const;

virtual int Read(char **text) const;

#endif

virtual int Write(ZIL_INT8 value) const;

virtual int Write(ZIL_UINT8 value) const;
```

ZIL_FILE

```
virtual int Write(ZIL_INT16 value) const;

virtual int Write(ZIL_UINT16 value) const;

virtual int Write(ZIL_INT32 value) const;

virtual int Write(ZIL_UINT32 value) const;

virtual int Write(ZIL_ICHAR *text) const;

virtual int Write(void *buffer, int size, int length) const;

#ifdef(ZIL_UNICODE)

virtual int Write(char *text) const;

#endif
```

Protected

Member variables

```
ZIL_FILE: :Z_ERROR error;
ZIL_ICHAR *pathName;
int access;
int handle;
unsigned mode;
```

Class description

Public members

Types, constants, and enums

UIS_READ

Sets file access read-only.

UIS_READWRITE

Sets file access read-write.

UIS_CREATE

When passed into the `ZIL_FILE` constructor, allocates a new file object.

If you use this flag to create a file object with a path name that doesn't already exist on disk, `ZIL_FILE` creates the file you specified. If you create a file object with a path name already on disk, `ZIL_FILE` overwrites the file.

If it could not open or create the file you specified, `ZIL_FILE` sets its handle to `-1` and returns `ERROR_ACCESS` as the error message.

UIS_OPENCREATE

When passed into the `ZIL_FILE` constructor, allocates a new file object and opens the file.

If you create a file object with a path name that doesn't already exist on disk, and if you pass in the `UIS_CREATE` flags, `ZIL_FILE` creates the file object you specified.

If `ZIL_FILE` could not open or create the file object you specified, it sets its handle to `-1` and returns `ERROR_ACCESS` as the error message.

UIS_TEMPORARY

Allocates a temporary file that is deleted when the file object is destroyed.

UIS_COPY

Copies a file to another file.

UIS_BINARY

Sets read/write mode to binary.

UIS_TEXT

Sets read/write mode to binary.

ZIL_FILE

ERROR_NONE

Indicates no error has occurred.

ERROR_NAME

Indicates you tried to read a file whose name wasn't found.

ERROR_NULL_STRING

Indicates you tried to open a file with a null filename.

ERROR_ACCESS

Indicates you tried to write to or read a file without proper permission.

SEEK_FROM_START

Seeks from the beginning of the file.

SEEK_FROM_CURRENT

Seeks from current location of the file pointer.

SEEK_FROM_END

Seeks from the end of the file.

Member functions

```
ZIL_FILE(const ZIL_ICHAR *pathName,
         UIS_FLAGS access = UIS_READ | UIS_BINARY);
```

Creates a new file object specified by pathName with read and write attributes specified by access. By default, attempts to open a read-only binary file.

```
virtual ~ZIL_FILE(void);
```

Destroys the file object and deallocates any memory it used. Does not, however, delete the file object unless you created the object as temporary.

```
long Length(void) const;
```

Returns the length of a file.

```
long Tell(void) const;
```

Returns the current position of the file pointer in a file object.

```
virtual ZIL_FILE::Z_ERROR Open(void);
```

Opens the file associated with the ZIL_FILE object.

```
ZIL_FILE *myFile;

myFile->open();
```

```
virtual ZIL_FILE::Z_ERROR Close(void);
```

Closes the file associated with the ZIL_FILE object..

```
ZIL_FILE *myFile;

myFile->Close( );
```

```
Z_ERROR Unlink(void);
```

Deletes the file associated with the ZIL_FILE object..

```
ZIL_FILE *myFile;

myFile->Unlink( );
```

ZIL_FILE

Z_ERROR Rename* const ZIL_ICHAR *pathName);

Renames a file to a new file specified by pathName.

```
ZIL_FILE *myFile;  
  
myFile->Rename( ZIL_ICHAR newPathname="newfile" );
```

Z_ERROR Seek(long offset, SEEK location) const;

Sets the position of the file pointer in a file object to a location that is offset bytes from a starting location in the file. Location can be either

- SEEK_FROM_CURRENT (the current location)**
- SEEK_FROM_START (the beginning of the file)**
- SEEK_FROM_END (the end of the file)**

```
ZIL_FILE *myFile;  
  
myFile->Seek(128232, SEEK_FROM_CURRENT);
```

virtual int Read(ZIL_INT8 *value) const;

Reads the data at the current file pointer into a ZIL_INT8 variable.
virtual int Read(ZIL_JJINT8 *value) const;

virtual int Read(ZIL_INT16 *value) const;

Reads the data at the current file pointer into a ZIL_INT16 variable.

virtual int Read(ZIL_INT32 *value) const;

Reads the data at the current file pointer into a ZIL_INT32 variable.

virtual int Read(ZIL_INT32 *value) const;

Reads the data at the current file pointer into a ZIL_INT32 variable.

virtual int Read(ZIL_ICHAR *text, int length) const;

Reads the data at the current file pointer into a string of a maximum length.

virtual int Read(ZIL_ICHAR **text) const;

Reads the data at the current file pointer into a n array of strings.

virtual int Read(void *buffer, int size, int length) const;

Reads the data at the current file pointer into a buffer of any type, up to a maximum length. Size is the size in bytes of the type specifier of the buffer. Default Read() function.

virtual int Read(char *text, int length) const;

Reads the data at the current file pointer into a string of a specific length (Unicode only).

virtual int Read(char **text) const;

Reads all strings in the file inside a file (Unicode only).

virtual int Write(ZIL_INT8 value) const;

Writes a ZIL_INT8 value in a file.

```
ZIL_FILE *myFile;
ZIL_INT8 number=127;

myFile->Write (number) ;
```

virtual int Write(ZIL_UINT8 value) const;

Writes a ZIL_INT8 value into a file.

virtual int Write(ZIL_INT16 value) const;

Writes a ZIL_INT16 value into a file.

virtual int Write(ZIL_INT16 value) const;

Writes a ZIL_INT16 value into a file.

virtual int Write(ZIL_INT32 value) const;

Writes a ZIL_INT32 value into a file.

ZIL_FILE

virtual int Write(ZIL_INT32 value) const;

Writes a ZIL_UINT32 value into a file.

virtual int Write(ZIL_ICHAR *text) const;

Writes a single- or double-byte string into a file.

virtual int Write(void *buffer, int size, int length) const;

Default Write() function. Writes the contents of an untyped buffer of a specific word size, up to a maximum length, into a ZIL_FILE object.

virtual int Write(char *text) const;

Writes a string of single-bytes into a file (Unicode only).

Protected

Member variables

ZIL_FILE::Z_ERROR error;

Error condition reported when reading or storing a file.

ZIL_ICHAR *pathName;

Pathname of the file.

int access;

The access attributes of the file—whether it is read-only or read-write,

int handle;

Handle to the file in a ZIL_FILE object. Assigned by the operating environment,

unsigned mode;

In operating environments with the POSIX API, mode represents the mode of the file. Default POSIX mode is 0664 (-rw-rw-r-). Non-POSIX operating environments set the mode to the access attributes of the file when it was created.

Addenda

This document contains additions and changes to the documentation shipped with OpenZinc 1.0.

Changes to General notebook page of Combo box

Since the documentation was printed, OpenZinc added some new features to the General page of the combo box notebook.

OpenZinc added a new field called List ID. This field provides access, not just to the presentation space of the list object, but to the pop-up list as well. List ID provides the ID number of the list. You can specify a new value, or you can view it to determine what list ID the Designer assigned to the list.

Changes to General notebook page of Pop-up item

Since the documentation was printed, OpenZinc added two new fields to the General page of the pop-up menu notebook.

- Menu name

Contains the name of the list of pop-up menu items associated with a parent items. Assigned automatically by OpenZinc Designer when a pop-up menu item has a list of child items attached. You can specify a new name for the menu list, or use the one assigned by OpenZinc Designer. (You will probably be able to identify it easier in the object directory or in the .HPP file if you specify a new name for it.)

- Menu ID

Represents the numberID of the pop-up item. Assigned automatically by OpenZinc Designer. You can specify a new value, or you can view it to determine what menu ID the Designer assigned to the menu.

Changes to the Geometry page of OpenZinc Designer

In release 1.0, OpenZinc updated the Geometry page of the information notebook. The new page style provides new ways to work with geometry management for all objects in the Designer.

In OpenZinc, an object can have three types of constraints:

- none
- absolute, and
- relative.

Further, the constraints work for the following sides:

- top
- bottom
- left, and
- right

In release 4.0, OpenZinc Designer visually represented constraints for each side of an object using four combo boxes connected to the top, bottom, left, and right of the object. In turn, each combo box contained a type of constraint, which was either no constraint or a relative or absolute constraint. To specify an object's constraint, you selected a constraint type from each combo box connected to the object's bitmap.

In release 1.0, OpenZinc simplified choosing constraints for each side of an object by replacing the visual representation of an object's constraints. To specify an object's constraint,

- select the side of the object from the list of sides in the combo box.
- Then select a type of constraint for that side by clicking the appropriate button.

OpenZinc replaced the visual representation of an object's constraints with a group called CONSTRAINT, which contains a combo box with the sides of an object, the three constraint types, and another combo box that lists potential anchors for sides of your object.

Combo box

The topmost combo box contains the four sides of the object in its list:

- LEFT
- TOP
- BOTTOM
- RIGHT

Each one of these sides can have its own type of constraint, and, depending on the constraint, its own anchor.

Constraints

In addition to a combo box, the CONSTRAINT group also has three buttons that represent the type of constraints you can assign to a particular side of the object. These constraints are:

- no constraint
- absolute

When using an absolute constraint, you can anchor the object to its parent or to one of its siblings with no restrictions.

When specifying an offset for an object that uses an absolute constraint, specify the offset in units of the current measurement system. You can determine the measurement system by turning to the Position page.

- relative

When using a relative constraint, you cannot specify an anchor for that side. When OpenZinc Designer detects a relative constraint, OpenZinc Designer automatically anchors it to the object's parent.

When determining an offset for an object that uses a relative constraint, specify the offset as proportion or percentage of distance from its parent.

Anchor

In addition to a combo box with the object's sides, and as well as three constraint buttons, the CONSTRAINT group also contains a combo box with a list of sibling and parent objects to which you can anchor the object. As mentioned above, when using a relative constraint, you cannot specify an anchor for that side. When OpenZinc Designer detects a relative constraint, OpenZinc Designer automatically anchors it to the object's parent.

New ability to access lists associated with objects

OpenZinc objects with attached lists, such as combo boxes or pop-up menu item options, have two components:

- the presentation space

The visual representation of the object. Contains all object components, including the list

- the list.

Contains the items or options of the object. Contained in the presentation space, but can be accessed separately from the presentation space using the list or menu ID or name specified in the General page.

New help system

In addition to improved help screens, OpenZinc 1.0's new help display system enables access to all help contents. To use it, press the Index button on a help context screen. It will bring up a new window that contains a vertical list with all help screens available in the application.

OpenZinc provides this new feature automatically. You don't have to do anything special to your OpenZinc 1.0 applications to enable this feature.

New inline functions for checking flags

Release 1.0 includes some new inline functions that will improve your ability to check the state or attributes of an object. These functions are written in plain English style to improve readability of your code.

With 4.0 and earlier releases, to check if an object's flag member was set with a certain flag, you used the `FlagSet()` or `FlagsSet()` macro. With release 1.0, you can use the same methods—or you can use the new inline functions instead. However, using the new functions will help improve code readability.

Using a new function to check the state or attributes of an object returns `TRUE` if the object has that state or attribute, and returns `FALSE` otherwise. Here are some examples of how you can use the new functions instead of the older `FlagSet()` and `FlagsSet()` macros.

Using new functions instead of `FlagSet()`

Using `FlagSet()`, you might have written code that looked like this:

```
UIW_TIME *myTime;

if (FlagSet(myTime->tmFlag, TMF_SYSTEM))
    //do something
```

In release 1.0, you can use the new functions to carry out the same check.

```
UIW_TIME *myTime;

if (myTime->DefaultsToSystem( ))
    // do something
```

Using multiple functions instead of `FlagsSet()`

Using `FlagsSet()`, you might have written code that looked like this:

```
UIW_TIME *myTime;

if (FlagsSet(myTime->tmFlag, TMF_SYSTEM | TMF_ZERO_FILL))
    //do something
```

In release 1.0, you can use multiple functions to carry out the same check.

```
UIW_TIME *myTime;

if (myTime->DefaultsToSystem( ) && myTime->HasZeroFill( ))
    // do something
```

Notice in the examples that

Addenda

- the code that uses the new flag-checking functions is more readable; and
- it's easier to remember how to accomplish a check.

The following is a list of new functions and the flags they're designed to affect.

TABLE 2. Flag-checking functions

Function name	Affected flag
<hr/>	
<i>Attachments</i>	ATCF_FLAGS
IsLeftAttachment()	ATCF_LEFT
IsTopAttachment()	ATCF_TOP
IsRightAttachment()	ATCF_RIGHT
IsBottomAttachment()	ATCF_BOTTOM
AppliesToOppositeSide()	ATCF_OPPOSITE
StretchesObject()	ATCF_STRETCH
<hr/>	
<i>Buttons</i>	BTF_FLAGS
AllowsToggling()	BTF_NO_TOGGLE
SelectsOnDownClick()	BTF_DOWN_CLICK
SelectsOnDoubleClick()	BTF_DOUBLE_CLICK
AutoRepeatsSelection()	BTF_REPEAT
AutoSizes()	BTF_AUTO_SIZE
Is3D()	BTF_NO_3D
IsCheckBox()	BTF_CHECK_BOX
IsRadioButton()	BTF_RADIO_BUTTON
SendsMessageWhenSelected()	BTF_SEND_MESSAGE
HasStaticArray()	BTF_STATIC_BITMAPARRAY
IsDefaultButton()	BTF_DEFAULT_BUTTON
<hr/>	
<i>Button status</i>	BTS_FLAGS
IsDepressed()	BTS_DEPRESSED
IsCurrentDefault()	BTS_DEFAULT
<hr/>	
<i>Dimension constraints</i>	DNCF_FLAGS
IsHeightConstraint()	DNCF_HEIGHT
IsWidthConstraint()	DNCF_WIDTH

TABLE 2. Flag-checking functions

Function name	Affected flag
<i>Dates</i>	
IsUS()	DTF_US_FORMAT
IsEuropean()	DTF_EUROPEAN_FORMAT
IsAsian()	DTF_ASIAN_FORMAT
IsMilitary()	DTF_MILITARY_FORMAT
HasDashSeparator()	DTF_DASH
HasSlashSeparator()	DTF_SLASH
HasAlphaMonth()	DTF_ALPHA_MONTH
UsesDayOfWeekFormat()	DTF_DAY_OF_WEEK
IsUpperCase()	DTF_UPPER_CASE
UsesShortYear()	DTF_SHORT_YEAR
UsesShortMonth()	DTF_SHORT_MONTH
UsesShortDay()	DTF_SHORT_DAY
HasZeroFill()	DTF_ZERO_FILL
DefaultsToSystem()	DTF_SYSTEM
<i>Icons</i>	
SelectsOnDoubleClick()	ICF_DOUBLE_CLICK
IsMinimizeIcon()	ICF_MINIMIZE_OBJECT
HasStaticArray()	ICF_STATIC_ICONARRAY
<i>Images</i>	
HasStaticHandle()	IMF_STATIC_HANDLE
IsAutoSize()	IMF_AUTOSIZE
IsScaled()	IMF_SCALED
IsTiled()	IMF_TILED
IsBackground()	IMF_BACKGROUND

Addenda

TABLE 2. Flag-checking functions

Function name	<u>Affected flag</u>
<i>Menu items</i>	
IsSeparator()	MNIF_SEPARATOR
IsMaximizeOption()	MNIF_MAXIMIZE
IsMinimizeOption()	MNIF_MINIMIZE
IsMoveOption()	MNIF_MOVE
IsSizeOption()	MNIF_SIZE
IsSwitchOption()	MNIF_SWITCH
IsRestoreOption()	MNIF_RESTORE
IsCloseOption()	MNIF_CLOSE
AllowsCheckMark()	MNIF_CHECK_MARK
SendsMessageWhenSelected()	MNIF_SEND_MESSAGE
IsSelectable()	MNIF_NON_SELECTABLE
IsAboutOption()	MNIF_ABOUT
<i>Numbers</i>	
HasCurrencySymbol()	NMF_FLAGS
HasCreditSymbol()	NMF_CURRENCY
IsCommaDelimited()	NMF_CREDIT
HasPercentageSymbol()	NMF_COMMAS
UsesScientificFormat ()	NMF_PERCENT
HasDecimalPoint()	NMF_SCIENTMC
	NMF_DECIMAL
<i>Relative constraints</i>	
IsLeftConstraint()	RLCF_FLAGS
IsTopConstraint()	RLCF_LEFT
IsRightConstraint()	RLCF_TOP
IsBottomConstraint()	RLCF_RIGHT
AppliesToOppositeSide()	RLCF_BOTTOM
StretchesObject()	RLCF_OPPOSITE
CentersObjectHorizontally()	RLCF_STRETCH
CentersObjectVertically()	RLCF_HORIZONTAL_CENTER
	RLCF_VERTICAL_CENTER

TABLE 2. Flag-checking functions

Function name	Affected flag
<i>Strings</i>	
HasVariableName()	STF_VARIABLE_NAME
IsLowerCase()	STF_LOWER_CASE
IsUpperCase()	STF_UPPER_CASE
IsPasswordStyle()	STF_PASSWORD
IsSubstringOfText()	STF_SUBSTRING
<i>System buttons</i>	
IsGeneric()	SYF_GENERIC
<i>Time</i>	
HasSeconds()	TMF_SECONDS
HasHundredths()	TMF_HUNDREDTHS
HasHours()	TMF_NO_HOURS
HasMinutes()	TMF_NO_MINUTES
Is12Hour()	TMF_TWELVE_HOUR
Is24Hour()	TMF_TWENTY_FOUR_HOUR
HasZeroFill()	TMF_ZERO_FILL
HasColonSeparator()	TMF_COLON_SEPARATOR
HasNullSeparator()	TMF_NO_SEPARATOR
IsUpperCase()	TMF_UPPER_CASE
IsLowerCase()	TMF_LOWER_CASE
DefaultsToSystem()	TMF_SYSTEM
<i>Storage</i>	
IsReadOnly()	UIS_READ
IsCreateMode()	UIS_CREATE
IsReadWrite()	UIS_READ_WRITE
IsCopyMode()	UIS_COPY
IsOpenCreateMode()	UIS_OPENCREATE
IsTemporary()	UIS_TEMPORARY
IsText ()	UIS_TEXT
IsBinary()	UIS_BINARY

TABLE 2. Flag-checking functions

Function name	Affected flag
<i>Lists</i>	
AllowsMultipleSelection()	WNF_SELECT_MULTIPLE
AutoSortsData()	WNF_AUTO_SORT
AllowsDragSelection()	WNF_CONTINUE_SELECT
SelectsOnDownClick()	WNF_AUTO_SELECT
HasWrappedData()	WNF_NO_WRAP
HasOwnerDrawChildren()	WNF_OWNERDRAW_CHILDREN
IsScrollable()	WNF_NO_SCROLL
<i>Window object actions</i>	
UsesOutsideRegion()	WOAF_OUTSIDE_REGION
IsNoncurrent()	WOAF_NON_CURRENT
IsTemporary()	WOAF_TEMPORARY
IsDestroyable()	WOAF_NO_DESTROY
UsesNormalHotKeys()	WOAF_NORMAL_HOT_KEYS
IsSizable()	WOAF_NO_SIZE
IsMovable()	WOAF_NO_MOVE
IsModal()	WOAF_MODAL
IsLocked()	WOAF_LOCKED
IsCopyDraggable()	WOAF_COPY_DRAG_OBJECT
AcceptsDrop()	WOAF_ACCEPTS_DROP
IsMoveDraggable()	WOAF_MOVE_DRAG_OBJECT
IsMDI()	WOAF_MDI_OBJECT
IsDialog()	WOAF_DIALOG_OBJECT
<i>Window objects</i>	
IsLeftJustified()	WOF_JUSTIFY_CENTER WOF_JUSTIFY_RIGHT
IsCenterJustified()	WOF_JUSTIFY_CENTER
IsRightJustified()	WOF_JUSTIFY_RIGHT
IsBordered()	WOF_BORDER
UsesUserData()	WOF_NO_ALLOCATE_DATA
IsViewOnly()	WOF_VIEW_ONLY
IsSupport()	WOF_SUPPORT_OBJECT
IsDefaultUnanswered()	WOF_UNANSWERED
IsDefaultInvalid()	WOF_INVALID

TABLE 2. Flag-checking functions

Function name	Affected flag
UsesAvailableRegion()	WOF_NON_FIELD_REGION
IsSelectable()	WOF_NON_SELECTABLE
IsAutoclear()	WOF_AUTO_CLEAR
UsesDefaultCellCoordinate()	WOF_MINICELL WOF_PIXEL
UsesDefaultMinicellCoordinate()	WOF_MINICELL
UsesDefaultPixelCoordinate()	WOF_PIXEL
<i>Window object status</i>	
HasConvertedCoordinates()	WOS_FLAGS
IsCurrent()	WOS_GRAPHICS
HasChanged()	WOS_CURRENT
IsSelected()	WOS_CHANGED
IsUnanswered()	WOS_SELECTED
IsInvalid()	WOS_UNANSWERED
IsMaximized()	WOS_INVALID
IsMinimized()	WOS_MAXIMIZED
IsUpToDate()	WOS_MINIMIZED
HasReadError()	WOS_REDISPLAY
IsOwnerDraw()	WOS_READ_ERROR
	WOS_OWNERDRAW

New mouse message

In 1.0, OpenZinc added a new mouse message entitled DM_DRAG, which has a value of -1214. This new message changes the mouse cursor to a drag image. Though your applications that use drag and drop already behave this way, you can now trap for this flag explicitly.

New open and print document events

Many graphical user interfaces enable the user to open and print documents by double-clicking and dragging their icons. To enable you to write OpenZinc applications that offer this ability, OpenZinc added two events:

- **S_OPEN_DOCUMENT**

Generated when the user double-clicks on one or multiple document icons, or when the user drags and drops a document onto an application icon. The system launches the application or makes it current if it's already running, then sends S_OPEN_DOCUMENT to the application. S_OPEN_DOCUMENT contains a pointer to a ZIL_ICHAR, which indicates the pathname of the document to be opened.

The system also generates S_OPEN_DOCUMENT when the user
—double-clicks the application icon

—launches the application without specifying a document name to open. For example, under Macintosh, the user may select the application icon from the Finder and chooses File | Open from the menu bar.

- **S_PRINT_DOCUMENT**

Generated when the user drags one or multiple document icons onto the printer icon. The system launches the application or makes it current, then sends S_PRINT_DOCUMENT to the application. S_PRINT_DOCUMENT contains a pointer to a ZIL_ICHAR, which indicates the pathname of the document to be printed.

Implementing opening and printing functionality

Remember, OpenZinc provides the potential for your applications to open and print documents—it does not provide the implementation. In order to enable your applications to open and print files, you must implement those features yourself. However, by implementing these features, your OpenZinc application will act like any other native application.

Implementing these features will enable the user to open a document by double-clicking on a document icon, even if the OpenZinc application associated with the document is inactive or is not currently running. Further, it will enable the user to print a document associated with a OpenZinc application by dragging the document icon to a printer icon without first launching the application.

Since OpenZinc is a multiplatform-enabled development environment, implementing these features will allow opening and printing a document to work in any graphical user interface that OpenZinc supports.

Restrictions on drag-and-drop under Macintosh

Under the Macintosh operating system, you can open a document by dragging and dropping it on the application icon only if the document's type is in the application's database of valid documents.

If you drag and drop a document on an application icon and the application can open the document type, the application icon will become highlighted. But if you try this with a document type not supported by the application, the application icon will not change.

Environment-specific mapping

OpenZinc objects can map system events to OpenZinc events. However, at present, the only environment in which OpenZinc maps system events to OpenZinc events is Macintosh, where OpenZinc maps Apple events into the corresponding S_OPEN_DOCUMENT and S_PRINT_DOCUMENT events. OpenZinc will provide additional mappings for other environments in future releases.

Using these events programmatically

This code snippet demonstrates a control structure for using S_OPEN_DOCUMENT and S_PRINT_DOCUMENT in a derived Window Manager class.

Program 1. Using the open and print document flags

```

EVENT_TYPE DERIVEDJrWINDOW_MANAGER::Event(const &event)
{
    if (event.type == S_OPEN_DOCUMENT)
    {
        if (event.text)
        {
            // Open the document specified by event.text
            *this + UIW_WINDOW::Generic(0, 2, 20, 6, event.text);
            delete event.text;
        }
        else
        {
            // Open a new document
            *this + UIW_WINDOW::Generic(0, 2, 20, 6, "Untitled");
        }
        return (event.type);
    }
    else if (event.type == S_PRINT_DOCUMENT)
    {
        // Print the document specified by event.text
        // Printing code goes here
        delete event.text;
        return (event.type);
    }
    else
        return (UI_WINDOW_MANAGER::Event(event));
}

```

Using S_OPEN_DOCUMENT

To use S_OPEN_DOCUMENT, employ the following approach.

1. Derive a class from a button or menu item that posts S_OPEN_DOCUMENT to the event queue.
2. Derive a class that contains a mechanism for opening a document from within your application when it receives S_OPEN_DOCUMENT. OpenZinc recommends you derive this class from UI_WINDOW_MANAGER; this will allow your application to run without modification under all platforms that OpenZinc supports.
3. Trap for S_OPEN_DOCUMENT in the derived object's Event() function. When Event() receives S_OPEN_DOCUMENT, it can use UI_PRINTER to print the document to the system printer.

4. After the event is processed, and if event.text has a value other than null, delete event.text to deallocate the memory the pathname used. (If it has a null value, the pathname took up no memory and need not be deleted.)

Using S_PRINT_DOCUMENT

To use S_PRINT_DOCUMENT, employ the following approach.

1. Derive a class from a button or menu item that posts S_PRINT_DOCUMENT to the event queue.
2. Derive a class that contains a mechanism for printing within your application. In this example, OpenZinc derives this class from UI_WINDOW_MANAGER to keep the opening and printing mechanism in the same class.
3. Trap for S_PRINT_DOCUMENT in the derived object's Event() function. When Event() receives S_PRINT_DOCUMENT, it can use UI_PRINTER to print the document to the system printer.

OpenZinc recommends that your mechanism for printing within your application print the document without opening it. This way, you can generalize the printing behavior of your application for all circumstances. For example, if your application prints the document without opening it, you can use the same mechanism for both when the user has an active document and when he doesn't. If your mechanism opened the document to print it, the user could find herself in the unexpected position of having *two* open copies of her document for the duration of the print job.

4. As with S_OPEN_DOCUMENT, delete event.text to deallocate the memory the pathname used.

New support for 3D controls in OpenZinc 1.0

Starting with OpenZinc 1.0, control objects can use the popular 3D look and feel. This feature is available only with OpenZinc applications running under Windows.

To use 3D controls in OpenZinc 1.0, do the following:

1. In UI_ENV.HPP, uncomment


```
#define ZIL_WINDOW_CTL3D
```

2. Rebuild the library.
3. In your program's makefile, add CTL3DV2 to the WIN_LIBS variable.
4. Make sure the include and lib path includes a path to the CTL3D.H header file and the control library. Also make sure CTL3D.DLL is in your path.

Functions moved from Designer to the library

OpenZinc moved the following functions from the Designer to the library in order to enable all OpenZinc programs to use their functionality.

- NormalizePosition()

```
static void NormalizePosition(  
    UI_WINDOW_OBJECT *object, // object with position to use  
    const UI_EVENT &event, // event.position to convert  
    UI_POSITION &position // converted position
```

Compares a coordinate's event.position to an object's trueRegion region.

Does this by converting event.position to a normalized coordinate.

This enables comparison with an object's trueRegion region (object->trueRegion).

- NormalizeString()

```
static void NormalizeString(  
    ZIL_ICHAR *destination, // destination string  
    const char *source // source string
```

Converts an eight-bit string to a 16-bit string, even if the 8-bit string is in Unicode.

- TrueToRelative()

```
static void TrueToRelative(  
    UI_WINDOW_OBJECT *object, // Object with coord to compare  
    const UI_POSITION &trueRegion, // Coord to position  
    UI_POSITION &relative // Converted position
```

Compares an object's relative coordinate with its relative region. Does

this by converting event's trueRegion coordinate to a relative coordinate.

Eliminates complications caused by nonfield-region objects and object contexts.

System events

All system events available to the user

The documentation does not clarify that all native system events are available to the OpenZinc programmer.

Some logical events generic to all platforms

The documentation does not clarify which logical events are generic to all platforms. The following is a list of logical events you can use on all platforms:

TABLE 3. Logical events generic to all platforms

Logical events

L_MDICHILD_EVENT	(ADD 500)
L_LOGICAL_EVENT	100
L_EXIT	1000
L_SELECT	1002
L_BEGIN_SELECT	1003
L_CONTINUE_SELECT	1004
L_END_SELECT	1005
L_HELP	1009
L_CANCEL	1010
L_EXIT_FUNCTION	1011
L_DOUBLE_CLICK	1012
L_MOVE	1013
L_SIZE	1014
L_PREVIOUS	1054
L_NEXT	1055
L_BEGIN_MARK	1101
L_CONTINUE_MARK	1102
L_END_MARK	1103
L_BEGIN_MOVE_DRAG	1150
L_CONTINUE_MOVE_DRAG	1151
L_END_MOVE_DRAG	1152
L_BEGIN_COPY_DRAG	1153

TABLE 3. Logical events generic to all platforms

<i>Logical events</i>	
L_CONTINUE_COPY_DRAG	1154
L_END_COPY_DRAG	1155
L_LOGICAL_LAST	9999

PowerPak 32 and crashing applications

If you build applications with the Borland PowerPak 32 DOS extender and without `UI_APPLICATION::Main()`, your applications could crash unexpectedly. When using PowerPak 32, your program can attempt to allocate memory for the stack while servicing a 16-bit real-mode interrupt, such as when the user moves the mouse. Due to a Borland bug, this will sometimes cause the application to crash.

To work around the problem, your program should allocate stack space while in 32-bit mode. For an example of how to do this, see the `CommitStack()` function in the OpenZinc source file `Z_APP.CPP`.

OpenZinc ships a patch for this problem on your distribution disks. You can find it in `OpenZinc\Powerpak`.

Errata

This document contains the errata to the documentation of OpenZinc Application Framework 1.0. Information in this document supersedes any information contained in manuals printed prior to release 1.0.

To use this document, scan the Table of Contents for the section in the documentation about which you have a question, then open the appropriate page. Or you can use the index to search for occurrences of a specific topic.

Programmer's Reference Volume One— Support Objects

Introduction

UIW_STRING inheritance

Although the class hierarchy indicates that UIW_STRING inherits from multiple parents, it is actually a normally derived object.

Chapter 1—UI APPLICATION

argc and argv

Contrary to the documentation of the constructor, argc and argv are available in Windows. In all environments, argc is always ≥ 1 , and argv[0] contains the program name.

Chapter 7—UI_DISPLAY

UI_DISPLAY::Text()

The description for left and top of UI_DISPLAY::Text() incorrectly states that they are relative to the region identified by the screenID passed in. Actually, left and top are relative to the coordinate system that is in turn relative to the trueRegion region passed into the function. (You can identify the trueRegion region by its screenID.)

Calling VirtualGet() and VirtualPut()

You must call VirtualGet() and VirtualPut() before calling any of the display primitives.

Chapter 12—UI_EVENT_MANAGER

DeviceState()

The manual erroneously stated in the description for DeviceState() that if the deviceType parameter is E_DEVICE, then each device's state will be set to deviceState. The manual should have stated that setting the parameter to E_DEVICE will affect only devices that exactly match the deviceType.

Chapter 14—UI_GEOMETRY_MANAGER

Pixel and minicell

You can specify the geometry-management constraints using minicell or pixel coordinates.

- WOF_MINICELL
Sets minicell coordinates.
- WOS_GRAPHICS
Sets pixel coordinates.

Chapter 17—UI_HELP_SYSTEM

Incorrect class name

The `UI_HELP_SYSTEM::SetLanguage` section is incorrectly entitled `UIW_HELP_SYSTEM:: SetLanguage`. (This error also occurs in the Table of Contents.)

Chapter 20—UI_LIST

Undocumented behavior of Add()

The documentation didn't explain all the functionality of the `UI_LIST::Add()` function.

Before adding an item to a list, `UI_LIST::Add()` and `UI_LIST::operator +` check to confirm that the element is not already in the list. If it is present, that item then becomes current.

When you add an element to a list, `Add()` makes that element current, whether or not it was previously a member of the list. In other words, if you have a list that contains the elements A, B, C, and you add element D to the list, element D will become current. And if you readd element A to the list, element A will become current

Chapter 22—UI_MACINTOSH_DISPLAY

maxColors

The member `maxColors` is actually an unsigned long instead of an int.

MapRGBColor()

The manual erroneously described the `MapRGBColor()` function as a static function. It should have described it as a virtual function.

patternTable and rgbColorMap

`UI_MACINTOSH_DISPLAY` no longer has the `patternTable` or `rgbColorMap` member variables.

PTN_RGB_COLOR

`UI_MACINTOSH_DISPLAY` now uses the `PTN_RGB_COLOR` identifier.

FontRec **fRec

All references to FontRec **fRec should now read FontInfo flInfo. flInfo is a Macintosh Toolbox structure that contains information about the font

PTN_INTERLEAVE_FILL

PTN_INTERLEAVE_FTLL is now used on the Macintosh. OpenZinc uses the QuickDraw 50% gray pen-pattern for this entry.

Chapter 32—UI_PRINTER

Dot-matrix printer support

In 1.0, a OpenZinc application running under DOS graphics or text mode is now able to print to an Epson-compatible nine- and 24-pin dot-matrix printer. This functionality is not available under other environments.

UI_PRINTER contains one member function specific to printing using a dot-matrix printer. It is

```
UI_PRINTER::PrintDotMatrixString(char *string)
```

Using this function prints the string passed in as a parameter.

One limitation of the UI_PRINTER class is that you cannot print graphics to a dot-matrix printer. OpenZinc is considering implementing this feature in a future release.

Using UI_PRINTER

To print documents with UI_PRINTER, use the following UI_PRINTER member functions:

- Begin all print jobs with `UI_PRINTER::BeginPrintJob()`.
- End all print jobs with `UI_PRINTER::EndPrintJob()`.
- Begin printing a page with `UI_PRINTER::BeginPage()`.
- End printing a page with `UI_PRINTER::EndPage()`.
- Place your drawing routines after `BeginPage()` and before `EndPage()`.
- Print all pages after `BeginPrintJob()` and before `EndPrintJob()`.
- `UI_PRINTER::ScreenDump()` calls `BeginPrintJob`, `EndPrintJob`, `BeginPage`, and `EndPage`. `ScreenDump()` is the only exception to the above rules.
- In DOS, you must create an environment variable `OpenZinc_PRINTER` and set the `lpt` identifier to a string that identifies the type of default printing and the printer port. If an environment variable doesn't exist, the class will write printer output to a PostScript file.

Types of default printing:

- PS
- PCL
- DM9
- DM24

Types of `lpt` values:

- LPT1
- LPT2
- LPT3

Examples:

```
SETOpenZinc_PRINTER=PS,LPT1
  Sets defaults to PostScript output, port 1.
```

```
SETOpenZinc_PRINTER=PCL,LPT2
  Sets defaults to PCL output, port 2.
```

```
SETOpenZinc_PRINTER=DM9,LPT3
  Sets defaults to nine-pin dot matrix, port 3.
```

```
SETOpenZinc_PRINTER=DM24,LPT1
  Sets defaults to 24-pin dot matrix, port 1.
```


When printing output to a PostScript file, the `OpenZinc_PRINTER` environment variable is ignored.

New `BeginPage()` function

After the documentation was printed, OpenZinc added a `BeginPage()` function:

```
virtual void BeginPage(int left, int top, int right, int bottom, int orientation = 0, int resolution = 0);
```

Use `BeginPage()` to print within a certain region and to print a document in landscape mode. Use `PRM_LANDSCAPE` for the orientation parameter to print landscape.

New `TextFormat()` function

After the documentation was printed, OpenZinc added a new member function `TextFormat()`. This member function formats text to fit on the page and inserts new lines and page breaks where necessary.

```
virtual void TextFormat(ZIL_SCREENID screenID, int x, int y, ZIL_ICHAR *text, const UI_PALETTE *palette, int width = -1, ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
```

For an example of how to use this function, read the source code to the `PRINTR` example.

CHAPTER 38—UI RELATIVE CONSTRAINT

Centering flags

The documentation incorrectly described the `RLCF_HORIZONTAL_CENTER` and `RLCF_VERTICAL_CENTER` flags, which center an object accurately in a window. It should have explained that flags cause the object's relative position to be calculated from the horizontal center and the vertical center of the object, instead of the left, right, top, or bottom of the object.

CHAPTER 42—UI WINDOW MANAGER

`exitFunction`

The documentation incorrectly defined the member variable `exitfunction`. The actual definition is:

```
EVENT_TYPE FunctionName(UI_DISPLAY *display, UI_EVENT_MANAGER *eventManager, UI_WINDOW_MANAGER *windowManager);
```

Chapter 43—UI_WINDOW_OBJECT

New member function and flag

After the documentation was printed, OpenZinc added the following member function and flag to UI_WINDOW_OBJECT.

- DrawFocus()

```
EVENT_TYPE DrawFocus(ZIL_SCREENID screenID,  
    UI_REGION &region, EVENT_TYPE ccode);
```

Draws a focus rectangle on the object.

- WOF_PIXEL

Causes the object's position and size parameters to be interpreted as pixel coordinates.

Chapter 53—ZIL_DECORATION_MANAGER

New defaultOSName

New member variable defaultOSName contains the ISO name of the current operating system.

Chapter 57—ZIL_I18N_MANAGER

New defaultOSName

New member variable defaultOSName contains the ISO name of the current operating system.

Chapter 61—ZIL_LANGUAGE_MANAGER

New defaultOSName

New member variable defaultOSName contains the ISO name of the current operating system.

Chapter 64—ZIL_LOCALE_MANAGER

New defaultOSName

New member variable defaultOSName contains the ISO name of the current operating system.

Programmer's Reference Volume Two— Window Objects

Introduction

Although the class hierarchy indicates that UIW_STRING inherits from multiple parents, it is actually a normally derived object.

Chapter 1—UIW_BIGNUM

Information requests

The documentation incorrectly describes the I_DECREMENT_VALUE and I_INCREMENT_VALUE requests for the Information() function. It states that the data parameter must be a pointer to an int. Instead, the parameter must be a pointer to ZIL_INT32.

NMF_SCIENTIFIC not supported

UIW_BIGNUM does not support the NMF_SCIENTIFIC flag.

Chapter 3—UIW_BUTTON

Toggle behavior

If a button is added to a vertical or horizontal list and if the BTF_NO_TOGGLE flag is on, it will have its BTF_NO_TOGGLE flag turned off.

Toggle appearance

The manual erroneously states that a toggle button usually appears flat if `BTF_NO_TOGGLE` is toggled to the selected state. Instead, the toggle button usually will appear depressed when toggled to the selected state.

TOP is bottom edge

Buttons are positioned with TOP as the bottom edge of the button.

Chapter 5—UIW_DATE

Information requests

The manual erroneously describes the `I_DECREMENT_VALUE` and `I_INCREMENT_VALUE` requests for the `Information()` function. It states that the data parameter must be a pointer to an int. Instead, the parameter must be a pointer to `ZIL_INT32`.

Chapter 14—UIW_POP_UP_ITEM

MNIF_ABOUT for Macintosh only

`MNIF_ABOUT` is currently ignored in all environments except Macintosh. On the Macintosh, a pop-up item created with this flag will be created as the About... option in the Apple menu. Please note that this is the preferred method for creating the Apple | About... option. The method documented in the Reference Manual will still work but might be removed in the future.

Chapter 19—UIW_REAL

NMF_DIGITS should be NMF_DECIMAL

The manual erroneously labeled a flag `NMF_DIGITS`. Its name is actually `NMF_DECIMAL`.

Chapter 20—UIW_SCROLL_BAR

Default width and height

A vertical scroll bar or slider created with a width of 0 will have a default width. A horizontal scroll bar or slider created with a height of 0 will have a default height

Chapter 22—UIW_STATUS_BAR

Adding objects to status bar

The documentation should explain that objects are added to the status bar the same way objects are added to a window.

Chapter 24—UIW_SYSTEM_BUTTON

Using Apple menu "About" item

The section describing how to create the Apple menu's "About" item on the Macintosh is no longer valid. While this method may still work, the preferred method is to use the new MNF_ABOUT flag, described in this document in the UIW_POP_UP_ITEM chapter section.

Chapter 25—UIW_TABLE

Building library with persistence

Currently, the library must be built with persistent load and store capability in order for the table object to function properly.

Chapter 29—UIW_TIME

Information requests

The documentation omitted a description of the I_DECREMENT_VALUE and I_INCREMENT_VALUE requests for the Information() function. See the description of these requests in the UIW_REAL chapter.

Chapter 33—UIW_WINDOW

New constructor

The UIW_WINDOW class has a new constructor and two new functions that allow it to use delta storage. The new constructor has the following signature:

```
UIW_WINDOW(const ZIL_ICHAR *name,  
            const ZIL_ICHAR *deltaName,  
            const ZIL_ICHAR *deltaPathName = ZIL_NULLP(ZIL_ICHAR),  
            ZIL_STORAGE_READ_ONLY *file = ZIL_NULLP(ZIL_STORAGE_READ_ONLY),  
            ZIL_STORAGE_READ_ONLY *deltaFile  
            =ZIL_NULLP(ZIL_STORAGE_READ_ONLY),  
            UI_ITEM *objectTable = ZIL_NULLP(UI_ITEM),  
            UI_ITEM *userTable = ZIL_NULLP(UI_ITEM));
```

The first new function has the following signature:

```
void DeltaLoad(const ZIL_ICHAR *name,  
               const ZIL_ICHAR *deltaName,  
               const ZIL_ICHAR *deltaPathName,  
               ZIL_STORAGE_READ_ONLY *file,  
               ZIL_STORAGE_READ_ONLY *deltaFile,  
               UI_ITEM *objectTable, UI_ITEM *userTable) ;
```

The second new function has the following signature:

```
void DeltaStore (const ZIL_ICHAR *name,  
                 const ZIL_ICHAR *deltaName,  
                 const ZIL_ICHAR *deltaPathName = ZIL_NULLP(ZIL_ICHAR) ,  
                 ZIL_STORAGE_READ_ONLY *file = ZIL_NULLP(ZIL_STORAGE_READ_ONLY),  
                 ZIL_STORAGE *deltaFile = ZIL_NULLP(ZIL_STORAGE),  
                 UI_ITEM *objectTable = ZILJSIULLP(UI_ITEM) ,  
                 UI_ITEM *userTable = ZIL_NULLP(UI_ITEM) ,  
                 int appendNames = FALSE);
```

Setting WOF_MDI_OBJECT

Setting **WOF_MDI_OBJECT** clears the **WOF_NON_FIELD_REGION** flag during construction.

Using default storage with the storage constructor

The documentation omitted an explanation of how to use the default storage when calling the storage constructor of UIW_WINDOW. Here's how to use default storage with UIW_WINDOW's storage constructor:

Errata

```
UIW_WINDOW *window =  
    new UIW_WINDOW("HELLO_UNIVERSE",  
        UI_WINDOW_OBJECT::defaultStorage);
```

This will load the window HELLO_UNIVERSE from the application's DAT file.

Chapter 35—ZAF MESSAGE WINDOW

New member variables

The ZAF_MESSAGE_WINDOW class now has three more member variables:

- static ZIL_ICHAR _className[];
- static int defaultInitialized;
- const ZIL_LANGUAGE *myLanguage;

These members serve the same purpose as the members in UIW_SYSTEM_BUTTON. See their description in that chapter.

Getting Started

Chapter 4—Writing Multiplatform Programs

Different contents than listed

The contents of some of the Macintosh libraries is different from what is listed. The following descriptions are accurate:

- Mac_ZIL2 Device and display classes
- Mac_ZIL3 Window Manager, window object, error classes
- Mac_ZIL8 Number classes
- Mac_ZIL9 Button and geometry management classes
- Mac_ZIL10 Controls classes

New libraries

There are also two new libraries:

- Mac_ZIL11—Window-derived classes
- Mac_ZIL12—Notebook and table classes

Also, the chapter erroneously calls the Macintosh application file an .EXE file.

Chapter 7—OpenZinc and C++

Misspelling

In both the Object creation section and the Object deletion section (pages 90, 95) UI_WINDOW_MANAGER is misspelled as UI_WINDOW_MANGER.

Chapter 18—Using Languages

Code to switch windows wrong

The code shown to switch windows when the user selects a new language is incorrect. The new code subtracts the original window in the Event() function rather than placing an S_SUBTRACT_OBJECT event on the queue. The new code adds the new window, then places the DELETE_OBJECT event on the queue.

CreateWin() function

The CreateWindow() function referred to in the Changing languages section is actually called CreateWin().

Chapter 19—Program Design

Accelerator keys work differently

The accelerator keys work differently than described in the manual. Previously, the accelerator keys worked only in DOS. They now work in all environments. The accelerator keys are now implemented as a custom event map table, mapped when looking for an accelerator key.

Appendix A—Compiler Considerations

New Mac_ZIL11

In addition to the Macintosh libraries listed, OpenZinc added a Mac_ZIL11 library.

Appendix B—Example Programs

Description of VALIDT program

The description of the VALIDT example program is inaccurate. Only one window appears.

The description of the ANALOG example program is inaccurate. It no longer has a status bar or a UIW_DATE object.

The PHONBK example refers to a `UI_STORAGE_OBJECT_READ_ONLY` member function. It is now called `ZIL_STORAGE_OBJECT_READ_ONLY`.

Appendix D—Keyboard and Mouse Mappings

Keyboard mappings for gray keys

All keyboard mappings for gray keys should be listed without '+'.

Backspace and Delete

The section for the Macintosh states that the `<Backspace>` key deletes the character to the left of the current cursor location. The `<Delete>` key is used for this purpose. The `<Delete Right>` key will delete the character to the right of the current cursor location.

`<Opt+Tab>`

On the Macintosh, `<Opt+Tab>` will move to the next window.

OpenZinc Designer

Chapter 1—Introducing OpenZinc Designer

Reactivating object on Mac

Because the Macintosh mouse only has one button, use <Option-click> to reactivate the last object added.

Pull-down menu on Mac

On the Macintosh the pull-down menu will not appear on the screen while editing a window. You can edit it through its parent window's Subobjects page.

Chapter 11—Edit Options

Grouping objects on Mac

To group objects using the Macintosh Designer, press <Option> and click somewhere in the window. Then you can draw a marquee around the objects.

Relative constraints in OpenZinc Designer

The documentation was unclear about how to set a relative constraint using OpenZinc Designer. The following is a step-by-step explanation of how to set a relative constraint in a new user interface using OpenZinc Designer.

1. Launch OpenZinc Designer.
2. Create a new window by selecting Window | Create.
3. Select the button object from the tool bar. Or, select Button from the Object | Control pull-down menu. When you have selected a button, the place object text field on the main window of OpenZinc Designer will display "Button."
4. Place the button on the window by clicking on the window.
5. **Once you have placed the button, double-click it to open its information notebook.**
6. Once you have opened the information notebook, select the Geometry notebook page.
7. Inside the Geometry notebook page, select Relative constraint.

8. Inside the Geometry notebook page and inside the CONSTRAINT group, click in the text field labeled Offset. Type in the amount of offset you wish.

When using relative constraints in OpenZinc Designer, offset is a percentage of distance from one border to its opposite border.

For example, if the button's parent window is 50 minicells wide, and if you type 40 in the Offset text field, OpenZinc Designer will place the button 20 minicells (40%) from the left border. If you enlarge the window to 100 minicells, OpenZinc Designer will place the button 40 minicells from the left border.

Chapter 15—Control Objects

Can't set Send flag for pull-down item

You can't set the Send user message flag for a pull-down item.

Chapter 17—Other Objects

There are several new features of UIW_TABLE.

Columns and RecordSize

The General Page of the table's Information notebook has two new fields.

- Columns

Specifies the number of columns that appear in the table. A table record will appear in each column.

- Record Size

The Record Size field specifies the size of the programmer's data structure used to store data for a single record. This value is used by the table, in conjunction with the number of records in the table, to determine how much memory to allocate for the table's data space.

Initializing table with DataSet()

After loading a window with a table on it in your application, call the table's DataSet() function to initialize the fields in the table.

Manipulating table visually

- You can size the table headers and table records visually by grabbing their border with the mouse and dragging it, just like with any other object. If a header or record is sized, the corresponding record or header should be sized appropriately so that the table headers and table records will scroll synchronously.
- You can add objects to the table records visually by dropping them on the record with the mouse, just like adding fields to a window.
- Double-clicking on a table header opens the table header's editor.
- Double-clicking on a table record opens the table record's editor.
- Double-clicking on an object in a table record opens the object's editor.

Window Object option in OpenZinc Designer

The documentation omitted an explanation of how to use OpenZinc Designer's Window Object option, which enables you to work visually with derived window objects. The Window Object option is located in the tool bar, as well as under Object | Other.

The following is a step-by-step explanation of how to use Window Object in a new application interface.

1. Launch OpenZinc Designer.
2. Create a new window by selecting Window | Create.
3. Select Window Object from the tool bar. (It's the last object in the tool bar.) Or, select Window Object from the Object | Other pull-down menu. When you have selected Window Object, the place object text field on the main window of OpenZinc Designer will display "Window Object."
4. Place the derived window object on the window by clicking on the window.
5. Once you have placed the derived object, double-click it to open its information notebook.
6. Once you have opened its information notebook, select the Advanced notebook page.
7. Inside the Advanced notebook page, click in the text field labeled DerivedName. Type in the name of your derived class.

8. Using the same derived class name you typed in the text field, derive a class within your source code the way you do normally. However, since your derived class will be loaded from a .DAT file, you must define a Load() function. Further, you must provide your class with a static New() function.

Chapter 19—Image Editor

Grouping objects

To group objects on the Macintosh, press <Option> and drag the mouse.

Appendix A—Compiling the Designer

Libraries

The manual mentions a Macintosh SCCPP700 Libraries folder. The folder is actually called SCCPP700 Library.

In addition to the Macintosh libraries listed in step 2, there is also a Window 4 library.

Defaults Editor

The manual refers improperly to the Defaults Editor as the I18N Defaults Editor in the Macintosh section.

Integrated OpenZinc Designer modules

At the time the documentation was printed, OpenZinc Designer running under the Macintosh was composed of different applications that you compiled separately. Currently, however, OpenZinc Designer under the Macintosh is an integrated application; that is, all of the editors have been placed into the same application. Access to any OpenZinc Designer editor on the Macintosh is accomplished through the Tools menu.

Drag and drop

OpenZinc Application Framework now supports drag and drop. Several new flags, events, and mouse states are required to use this capability. In addition to these values, described below, you must compile the library with ZIL_STORE and ZIL_LOAD defined in UI.ENV.HPP in order for drag and drop to work.

Errata

The *Reference* manual lists the `WOAF_DRAG_OBJECT` in several places; `WOAF_MOVE_DRAG_OBJECT` and `WOAF_COPY_DRAG_OBJECT` replaced it after the manual went to press.

Here is a list of the flags you can use with drag-and-drop:

- `WOAF_ACCEPTS_DROP`
Indicates that the object can accept a drop operation.
- `WOAF_MOVE_DRAG_OBJECT`
Indicates that the object can be moved in a drag and drop operation.
- `WOAF_COPY_DRAG_OBJECT`
Indicates that the object can be copied in a drag and drop operation.
- `DM_CANCEL`
Indicates that a drop may not be performed at the present mouse location.
- `DM_DRAG_MOVE`
Indicates that a drop at the present mouse location would result in a move operation.
- `DM_DRAG_COPY`
Indicates that a drop at the present mouse location would result in a copy operation.
- `DM_DRAG_MOVE_MULTIPLE`
Indicates that a drop at the present mouse location would result in a move operation.
- `DM_DRAG_COPY_MULTIPLE`
Indicates that a drop at the present mouse location would result in a copy operation.
- `S_DRAG_MOVE_OBJECT`
Sent to an object when a move operation is performed over the object. The object sets the mouse cursor to an appropriate state.
- `S_DRAG_COPY_OBJECT`
Sent to an object when a copy operation is performed over the object. The object sets the mouse cursor to an appropriate state.

- **S_DRAG_DEFAULT**
Sent to an object when a drag operation is in effect but neither a copy or move has been specified. The object receiving this event determines the type of drag and sets the mouse cursor appropriately.
- **S_DROP_MOVE_OBJECT**
Sent to an object to indicate that a move drop was just performed on the object.
- **S_DROP_COPY_OBJECT**
Sent to an object to indicate that a copy drop was just performed on the object.
- **S_DROP_DEFAULT**
Sent to an object when a drop operation is in effect but neither a copy or move has been specified. The object which is receiving this event then determines the type of drag and sets the mouse cursor appropriately.
- **L_BEGIN_MOVE_DRAG**
Indicates that a move-drag operation was just started with a mouse down-click.
- **L_CONTINUE_MOVE_DRAG**
Indicates that a move-drag operation is continuing with the mouse being dragged while the mouse button is depressed.
- **L_END_MOVE_DRAG**
Indicates that a move-drag operation was just completed with a mouse up-click.
- **L_BEGIN_COPY_DRAG**
Indicates that a copy-drag operation was just started with a mouse down-click.
- **L_CONTINUE_COPY_DRAG**
Indicates that a copy-drag operation is continuing with the mouse being dragged while the mouse button is depressed.
- **L_END_COPY_DRAG**
Indicates that a copy-drag operation was just completed with a mouse up-click. If an object with the `WOAF_ACCEPTS_DROP` flag set receives an `S_DRAG_*` or an `S_DROP_*` message, it will call the receiving object's user function, passing it the `S_*` message it

Errata

received. The user function can process the message, if desired. The user function must return one of the following values when it receives a drag or drop message:

- S_ERROR

Indicates that the dragged object cannot be dropped here.

0 indicates that the user function processed the message.

- S_UNKNOWN

Indicates that the user function did not process the message and that the receiving object should handle the message.

Mac and NEXTSTEP File and Edit menus

To allow maximum flexibility in your OpenZinc applications running under Macintosh and NEXTSTEP, OpenZinc recommends that you use the following menus and items and implement the following functionality:

- File

- New

Create a new document.

- Open

Open an existing document.

- Close

Close the current document.

- Print

Print the current document.

- Quit

Quit the application.

- Edit

- Cut

Cut the selected text or object from the document and place in the paste buffer (Clipboard in Mac, Pasteboard in NEXTSTEP).

- Copy

Copy the selected text or object to the paste buffer.

—Paste

Paste text or an object from the paste buffer to the current cursor location in the document.

Microsoft and Watcom graphics

In DOS, if you are using the `UI_MSC_DISPLAY` with the Microsoft compiler, or the `UI_WCC_DISPLAY` with the Watcom compiler, put `HELVB.FON` somewhere in the path so that the fonts can be loaded.

New virtualized `Add()` and `Subtract()` functions

For 1.0, OpenZinc has modified the `Add()` and `Subtract()` member functions of some classes derived from `UIW_WINDOW`. Before OpenZinc 1.0, `Add()` and `Subtract()` were defined in the `UI_LIST` class; adding and subtracting elements to and from a list was accomplished by an instance of `UI_LIST`. In OpenZinc 1.0, `Add()` and `Subtract()` are virtual functions, so that objects derived from `UI_LIST` can implement new adding and subtracting functionality specific to those objects.

These newly virtualized functions will have no effect on classes derived from `UI_LIST` when upgrading existing OpenZinc applications to 1.0.

The classes with overloaded `Add()` and `Subtract()` functions are:

- `UIW_VT_LIST`
- `UIW_HZ_LIST`
- `UIW_COMBO_BOX`
- `UIW_STATUS_BAR`
- `UIW_PULL_DOWN_MENU`
- `UIW_POP_UP_MENU`

Recommended stack size

The following are recommended stack sizes for use with OpenZinc for applicable environments and compilers.

TABLE 4. Recommended stack size by environment and compiler

Environment	Compiler	Size
DOS	Watcom	48K
	Symantec (except 32-bit)	20K
	Symantec (32-bit)	16K
	Borland (real, PharLap 16-bit)	20K
	Borland (PowerPack 16, Rational DOS 16M 16-bit)	18.75K
	Borland (PowerPack 32) default	1024K
	Microsoft 1.0 (PharLap TNT 6.1 32-bit) default	32K
	Microsoft 1.5 (real)	5K
	Microsoft 1.5 (PharLap 16-bit, DOS 16M 16-bit)	12K
Windows	All compilers	18.75K
OS/2	All compilers	94K
Motif (DESQview/X)	Watcom	48K
Motif (QNX)	Watcom	48K
Power Macintosh	All compilers	64K

Macintosh heap size

A minimum heap size of 1024K is recommended for any Macintosh application.

OpenZinc Technical Notes

This section contains the Technical Notes for OpenZinc Application Framework 1.0. The Technical Notes are brief technical explanations of how to accomplish various programming tasks with OpenZinc. If you are having problems using OpenZinc, scan the Technical Notes for a topic concerning your problem. You can often find a note describing how to solve the problem, thereby helping you avoid placing a call to OpenZinc Technical Support.

Section 1

IDEs

1.1 Working with the Borland IDE

ZD-TN1000

Keywords: Borland IDE
Version: 3.6 R1 and later
Component: Borland IDE
Platforms: DOS, Windows
Issued: August 1, 1994

Question

What do I need to do in Borland's 4.0 IDE to make it work properly with OpenZinc?

Answer

Set up your project files first. Make sure you include **.CPP** files first, then OpenZinc **.LIB** files as required for the particular environment, then **.DEF** and **.RC** for Windows programs.

Be careful that the project hierarchy is correct. Borland has changed the way that projects are set up in the IDE of Borland 4.0. The library and code files should come directly under the target executable unless one of the **.CPP** files is dependent upon another.

For example:

Correct	Incorrect
<pre>-prog.exe [target] I-prog.cpp - sub.cpp - \OpenZinc\lib\btcp400\dos_zil.lib I- \OpenZinc\lib\btcp400\dos_gfx.lib - \OpenZinc\lib\btcp400\bc_lgfx.lib</pre>	<pre>-prog.exe [target] - prog.cpp I- sub.cpp I- \OpenZinc\lib\btcp400\dos_zil.lib - \OpenZinc\lib\btcp400\dos_gfx.lib - \OpenZinc\lib\btcp400\bc_lgfx.lib</pre>

Make sure you have set the proper target. This is usually an **.EXE** file.

The only libraries that should be used in the IDE are the **RUNTIME** libraries. Don't use the **GRAPHICS** (unless you are using the **BGI** graphics instead of the **GFX**), or the **OWL**, or **CLASS** libraries.

Make sure you turn off all exception handling and RTTI options. The reason for this is that OpenZinc libraries are not compiled with exception handling or RTTI information. If you want this information you can recompile the libraries with exception handling and RTTI information. This will, however, dramatically increase the size of your executable.

Make sure the directories are correct for the include and library files. These should include Borland's **include** and **lib** as well as OpenZinc's **include** and **lib** directories.

Select the large memory model.

1.2 Setting up the Microsoft IDE to work with OpenZinc ZD-TN1006

Keywords: IDE, Microsoft, Visual C++
 Versions: All
 Components: Microsoft IDE
 Platforms: DOS, Windows
 Issued: August 1, 1994

Question

How do I set up the Microsoft IDE so that it works properly with OpenZinc?

Answer

The Microsoft IDE requires that scanning dependencies in the Workbench be resolved. Two methods can be used to accomplish this. (The first method is recommended since it will eliminate the need for changes as OpenZinc is updated.)

Either add the following filenames to the end of your **MSVC\BIN\SYSINCL.DAT** file:

- **KEYSYM.H, XM.H, TIMER.H, INTRINSI.H, DPSNEXT.H, EVENT.H**, and possibly other **.H** references that Microsoft Workbench doesn't recognize.

Or comment out the following lines in the OpenZinc header files:

- OpenZinc 4.0: lines 652, 667 in **UI_ENV.HPP** and 641 in **UI_MAP.HPP**.
- OpenZinc 3.6: lines 641, 642 in **UI_ENV.HPP** and 630 in **UI_MAP.HPP**.
- OpenZinc 3.5: lines 477, 478 in **UI_ENV.HPP** and 598 in **UI_EVT.HPP**.

Change the settings inside Microsoft Workbench as follows:

- **Project | New** and set your **Project** type to **MS-DOS** or **Windows Application** (.EXE)
- Add your **.CPP** files.
- For DOS only:
 - Add the **DOS_ZIL.LIB**.
 - Add **DOS_GFX.LIB** or **DOS_MSC.LIB**.
 - Add **MS_LGFX.LIB** if you use **DOS_GFX.LIB**.
- For Windows only:
 - Add the **WIN_ZIL.LIB**
- Under **Options | Project...**
 - Choose the **Compiler... Button**

Setting up the Microsoft IDE to work with OpenZinc ZD-TN1006

- Under **Code Generation** set **CPU** to **8086/8088**
- Under **Memory Model** set **Model** to **Large**
- Choose the **Linker... Button**
- Under **Input**, turn on **Prevent Use of Extended Dictionary**
- Under **Memory Image**, set **Max Number of Segments** to **256** (OpenZinc **4.0** only)
- For DOS only:
 - Under **Input**, add **".graphics"** to **Libraries**
 - For Windows only:
- Under **Memory Image**, remove entry in **Stack Size** if using *STACK* in your **.DEF** file.

1.3	Watcom 10.0 IDE for Windows	ZD-TN5006
Keywords:	Watcom IDE, Windows	
Versions:	4.0	
Components:		
Platforms:	Windows	
Issued:	October 13, 1994	

Question

How do you build a OpenZinc application for Windows using the Watcom 10.0 IDE?

Answer

To build a OpenZinc application for Windows using the Watcom 10.0 IDE, do the following:

Create a new project by selecting **File I New project** (e.g., **whello.prj**).

Enter the desired target name (e.g., **whello.exe**).

Name the source files needed to build the target by selecting **Source I New source** (e.g., **helloworld.cpp c:\OpenZinc\lib\wccpp\win_zilib whello.def whello.rc**).

Choose **Options I C++ Compiler switches...** and make the following modifications:

File Option Switches

- Include directories—should be amended to also include `c:\OpenZinc\include`

Diagnostic Switches

- Warning level—should be set to Warning level 4[-w4]

Optimization Switches

- Style of optimization—should be set to Space optimizations [-os]
- Relax alias checking[-oa]—should be turned on

CodeGen Option Switches

- Constants in code segment[-zc] - should be turned on

Memory Model and Processor Switches

- Memory Model—should be set to Large model [-ml]

Choose **Options I Windows Linking Switches...** and make the following modifications:

Watcom 10.0 IDE for Windows ZD-TN5006

Basic Switches

- Other options[,]
—should be set to op heapsize=28k
- Stack: [op st]
—should be set to 16k

Resource Switches

- Requires Windows 3.0 or later[-30]
—should be turned on

Make the target by selecting **Targets I Make**.

Run the target by selecting **Targets I Run**.

Section 2

Fonts

Abstract: Changing object fonts
Keywords: font, *fontTable*
Version: 3.0 and later
Component: Library
Platforms: All
Issued: August 1, 1994

Question

How do I change a font for an object?

Answer

Each object derived from **UI_WINDOW_OBJECT** has a member variable called font. This member is an index into the font table. The font table is an array of fonts. The size of the array is 10. The *fontTable* variable is a member of the display class that you have chosen to use such as **UI_GRAPHICS_DISPLAY**. It is a protected member of the class and is not directly accessible. There are two ways to modify this variable.

You can derive your own object and change the font value in the constructor of your object such as:

```
MY_OBJECT::MY_OBJECT(...) : UI_WINDOW_OBJECT(...)  
{  
    font = 3;  
}
```

This will cause the object to use entry 3 in the font table.

Version 3.6 introduced a new public member function of the **UI_WINDOW_OBJECT** class. This function is called **Font(LOGICAL_FONT _font)**, and takes an integer value as a parameter. The same thing is accomplished as in the previous example by making the following call.

```
object->Font(3);
```

This function can be used by any valid object pointer. Typically the use of this function would be for objects that have viewable text associated with them, such as strings, prompts, buttons, etc.

Notes:

For additional information on changing/adding fonts, see the technote on creating and adding a new font. There are also system specific technotes available.

2.2 Adding a new font to the *fontTable* under OS/2

ZD-TN1002

Keywords: OS/2, font, *fontTable*
 Version: 3.5 and later
 Component: Library
 Platforms: OS/2
 Issued: July 18, 1994

Question

How do I add a new font to the *fontTable* under OS/2?

Answer

To load a font, use specific OS/2 system commands found in the OS/2 code file 0_DSP.CPP. The following is an extract from this file.

Get a pointer to the desktop.

```
HPS desktopPS = WinGetPS(HWND_DESKTOP);
```

Get the number of available fonts from the system.

```
LONG maxFonts = 0;
maxFonts = GpiQueryFonts(desktopPS, QF_PUBLIC, (PSZ)"Courier", &maxFonts,
    sizeof(FONTMETRICS), NULLP(FONTMETRICS));
```

Create the space for the new font.

```
FONTMETRICS *font = new FONTMETRICS[maxFonts];
```

Load the available fonts into the newly created font.

```
GpiQueryFonts(desktopPS, QF_PUBLIC, (PSZ)"Courier", &maxFonts, sizeof(FONTMETRICS),
    font);
```

Add the font, based on 100 point, into the *fontTable* so it is available for use.

```
for (int i = 0; i < maxFonts; i++)
{
    if (font[i].sNominalPointSize <= 100 &&
        !UI_OS2_DISPLAY::fontTable[MY_FONT_ENTRY].IMatch)
    {
        UI_OS2_DISPLAY::fontTable[MY_FONT_ENTRY] = font[i];
        break;
    }
}
```

Free the space.

```
delete font;
WinReleasePS(desktopPS);
```

2.3 Adding a new font to the *fontTable* in GFX graphics ZD-TN1003

Keywords: font, *GFX*, *fontTable*
Versions: All
Components: GFX Graphics, **UI_GRAPHICS_DISPLAY**
Platforms: DOS
Issued: July 29, 1994

Question

How do I create and add a new font to *the fontTable* using GFX graphics?

Answer

Create the font using one of two methods.

The first method is to use Borland's font editor to create the font. When you save it, the font editor will produce a **.CHR** file. Then use the **BGI2GFX** program to convert the **.CHR** file to a **.CPP** file. This file can then be linked into your code. If the utility is not compiled, the code for **BGI2GFX** is located in **\OpenZinc\SOURCE\CONVERT** subdirectory and can be readily compiled with the existing makefile.

The second method is to get the GFX font editor from C-Source, the developer of GFX, or get their fonts. These can be readily linked into your source. C-Source may be contacted at 616-725-4882.

Make sure that the following declarations have been added to the bottom of your newly created **.CPP** font file. There should be a structure definition as well as a function that returns a pointer to the newly created font. The following is an extract from the **ISO_DIAL.CPP** file located in your **\OpenZinc\SOURCE\GFX** subdirectory.

- Structure definition of the new font:

```
static FONT iso_dial
{
    iso_dial_offset_table, iso_dial_data_table, 176, 13, 0, 255, 12, 0, 11, 0
};
```

- Function that returns the pointer to the new font:

```
FONT *GFX_iso_dial_font (void)
{
    return (&iso_dial);
}
```

Next, declare in your main **.CPP** file or header file some external reference to the same functions that are located in the bottom of your font's **.CPP** file. The following is an example of the type of declaration that should be used to return a pointer to your new font. The code sample is taken from **D_GFXDSP.CPP**.

```
#define GFX_DIALOG_FONT GFX_iso_dial_font
```

```
extern FONT *GFX_DIALOG_FONT(void);
```

Finally, load the font into the font table for use. The member, *fontTable*, is a public member of the display class that you are using, such as **UI_GRAPHICS_DISPLAY**, and can contain up to ten fonts. Loading the font must be done after your display has been constructed and before you create your objects. If you are using **UI_APPLICATION** then we suggest doing it at the top of your code since it constructs your display. That way the font will be loaded into the fontTable where you can use it as you load your various resources. The following is an extract from **D_GFXDSP.CPP**.

- Initialize a new font variable for use.

```
GRAPHICSFONT font = { 0, 0, 0 };
FONT *tmp;
```

- Load and insert the new font into the fontTable.

```
font.font = OpenMemFont(tmp = GFX_DIALOG_FONT());
if (font.font >= 0 && _loadFont)
{
    font.maxWidth = tmp->widest_cell - 1;
    font.maxHeight = tmp->height - 1;
    // Locations 0..9 are possible.
    UI_GRAPHICS_DISPLAY::fontTable[FNT_DIALOG_FONT] = font;
}
```

Notes:

- The sample code shown here is used to load the dialog font into the *fontTable*. This is to give you something tangible to look at when you are trying to load your own. The files that are referred to in this technote are ISO_DIAL.CPP and D_GFXDSP.CPP and should be consulted.

2.4 Adding a new font to the *fontTable* in Motif

ZD-TN2001

Keywords: font, Motif
 Versions: 3.5, 3.6, 4.0
 Components: Library
 Platforms: Motif
 Issued: July 14, 1994
 File: MJFONTS.TAR

Question

How do I change or add a font in Motif?

Answer

The first three entries of the font table are initialized by OpenZinc in **M_DSP.CPP**. Fonts are loaded from default strings in the OpenZinc App.**Xdefaults** file. If the **.Xdefaults** file is not found, or the font strings are not found in the **.Xdefaults** file, there are default strings in **M_DSP.CPP** that are used as the default. The first three fonts can be changed at run time by modifying the ZincApp **.Xdefaults** file. For example, to change the small font used by OpenZinc the small font entry could be changed from:

```
*zincSmallFont: *-helvetica-medium-r-*--12-*--iso8859-1
```

to the following for a italic small font.

```
*zincSmallFont: *-helvetica-medium-o-*--14-*--iso8859-1
```

Fonts can also be added in code. If a new font is added to the font table there should also be added an entry in the **.Xdefaults** file for that font. The following code is necessary for adding an italic small font to the font table.

```
const int FNT_ITALIC_SMALL= 3;
// Create pointer for string in XResource data base,
typedef struct {
    char *italicSmallFont;
} AppData;
#define XmNitalicSmallFont"italicSmallFont"
#define XmCItalicSmallFont"ItalicSmallFont"
// Create XResource sturcture
static XtResource resources[] =
{

    XmNitalicSmallFont,
    XmCItalicSmallFont,
    XmRString,
    sizeof(char *),
    XtOffsetOf(AppData, italicSmallFont),
```

Adding a new font to the fontTable in Motif ZD-TN2001

```
    XmRString,
// If the .Xdefaults file or the font string is not found this font will be used.
    (XtPointer)"-*-helvetica-medium-o-*--12-*--iso8859-1"
    },
};
// Get the font name from X and load it.
AppData appData;
XtGetApplicationResources(display->topShell, (XtPointer)&appData,resources,
    XtNumber(resources), NULLP(Arg), 0);
display->fontTable[FNT_ITALIC_SMALL].fontStruct =
    XLoadQueryFont(display->xDisplay, appData.italicSmallFont);
// If using OpenZinc's UnicodeKey
#ifdef ZIL_UNICODE
// Create font set and load font for Unicode
XmFontListEntry entry;
XmFontType type;
entry = XmFontListEntryLoad(display->xDisplay, appData.italicSmallFont,
    XmFONT_IS_FONTSET,XmFONTLIST_DEFAULT_TAG);
display->fontTable[FNT_ITALIC_SMALL].fontList = XmFontListAppendEntry (NULL,
    entry);
display->fontTable[FNT_ITALIC_SMALL].fontSet =
    (XFontSet)XmFontListEntryGetFont(entry, &type);
XmFontListEntryFree(&entry);
#else
display->fontTable[FNT_ITALIC_SMALL].fontList =
    XmFontListCreate(display->fontTable[FNT_ITALIC_SMALL].fontStruct,
    XmSTRING_DEFAULT_CHARSET);
#endif
}
```

The following line should be added to the .Xdefaults file:

```
•italicSmallFont:-*-helvetica-medium-o-*--14-*--iso8859-1
```

Note:

- Other platforms use different methods to load new fonts into the font table. See technotes for those platforms for specific information.
- There is a program on our BBS and ftp server that uses the above code to add more fonts to the font table. On the BBS the file is in the 3.6 Conference under User Contributions and is called M_FONTS.TAR. On the Internet the file is in PUB/CONTRIB/36 and is called M_FONTS.TAR.Z. This example also contains an example of the .Xdefaults file.

Adding a fixed-width font to the `fontTable` under Windows ZD-TN5002

2.5 Adding a fixed-width font to the `fontTable` under Windows ZD-TN5002

Keywords: fixed-width font, `fontTable`, Windows

Versions: 3.0 and later

Components: Library

Platforms: MS Windows

Issued: August 1, 1994

Question

How do you add a fixed-width font to *the* `fontTable` under MS Windows?

Answer

Adding a fixed-width font to the `fontTable` under MS Windows is straightforward. The following code shows how this is done.

```
// Initialize the display.
UI_DISPLAY *display = new UI_MSWINDOWS_DISPLAY(hInstance, hPrevInstance, nCmdShow);

// Get a handle to a fixed width font.
HFONT fixedFont = GetStockObject(ANSI_FIXED_FONT);

// Assign the fixed-width font to the 4th entry (index 3) of the fontTable.
UI_MSWINDOWS_DISPLAY:.-fontTable[3] = fixedFont;

// Assign the appropriate index to a window object,
object->Font(3);
```

The font ID passed to `GetStockObject()` could also be `OEM_FIXED_FONT` or `SYSTEM_FIXED_FONT`. See your MS Windows reference guide for additional information concerning stock fonts.

Section 3

Event flow, messaging, and flags

3.1 Event flow

ZD-TN1005

Keywords: event, message passing
Version: All
Component: Library
Platforms: DOS, OS/2, Windows, Macintosh
Issued: August 22, 1994

Question

How do events flow through the system?

Answer

OpenZinc uses three different methods for event flow in order to maintain full compatibility with operating systems. The three methods are top-down, bottom-up, and NEXTSTEP. This technote explains the first two.

Top-Down (DOS, Macintosh, Curses):

After the event is placed in the event queue, the Event Manager pulls the event from the queue and dispatches it to the window manager. At that point the Window Manager has three options:

- Act on the event—as in the case of *S_CLOSE*;
- Dispatch the message to the current window; or
- Make an object (a window) current, then dispatch the message.

If a window receives the event, it then has the same three options:

- Act on the event—as in the case of *L_NEXT* (the <tab> key);
- Dispatch the message to the current object; or
- Make an object current, then dispatch the message (mouse click on a button).

If an object receives the event it has two options:

- Act on the event; or
- Pass the message to its base class (usually *UI_WINDOW_OBJECT*) and let it handle the event.

If the object still doesn't know what to do with the message then *S_UNKNOWN* is returned. At this point the parent object may pass the message to its base class. The message is propagated back to the window manager in this manner, and if nothing is done with the message, it is then discarded.

Bottom-Up (Windows, OS/2, Motif):

In the case of bottom-up event flow, the event manager gets the event and gives the event to the window manager. At that point the window manager has two options:

- Pass the event to the operating system if the events are native system messages (most); or
- Pass the message as described in top-down (OpenZinc-specific messages).

The operating system event handler—OpenZinc calls it the "black box"—then passes the event to the current object where it has the same two choices described for top-down:

- Act on the event; or
- Pass the message to its base class (usually **UIW_WINDOW_OBJECT**) and let it handle the event.

If the message is not used by the object then the message is given back to the operating system where its default callback function is called.

Notes:

- Top-down event flow may seem like a long route for a message to follow, but very few messages ever get this far, so overhead incurred by this flexible architecture is minimal.
- All OpenZinc-specific messages are passed as described in the top-down section. Mouse and keyboard messages are all system specific.
- NEXTSTEP event flow resembles bottom-up, except that the event is not handled first by the OpenZinc Event Manager. OpenZinc 'sees' the event for the first time at the object level and handles or passes the event from there.

3.2 Local control loops

ZD-TN1008

Keywords: control, main loop, local, modal
Versions: All
Components: Library
Platforms: All
Issued: November 17, 1994
File: ctloop.zip

Question

How do I determine when to write a local control loop?

Answer

Local control loops can cause of difficult problems and are not recommended unless absolutely necessary. If not implemented properly, the system may become confused because it doesn't know the origin or destination of events.

There are few instances require a local control loop. The only reason to have a local control loop is if the user's input is necessary for the completion of a currently running task. The error system falls in this category. Another use could possibly be for to allow the user to break out of a running process. Regardless, you must design tasks to work in "pieces," thereby preserving the event-driven characteristics of graphical user interfaces.

A local control loop is used to take complete control over the trapping and dispatching of events. This means that all normal event flow is halted. If the programmed loop doesn't responsibly take control then other events could slip through, thereby causing confusion in the system.

For a good example of a local control loop you should examine the DOS error system. On DOS it was necessary for OpenZinc to create its own error system since DOS does not provide one. On other operating systems is able to call native controls to handle the task (e.g. Windows calls **MessageBox()**, OS/2 calls **WinMessageBox()**, etc.) These environment-specific objects wait for the user to select one of the buttons on the window, then allows OpenZinc to test the return value and respond accordingly.

The control code in this technote was taken from the DOS **ErrorMessage()** function in **D_ERRORI.CPP** and modified for the purpose of this example. The following snippet shows one case statement from a derived window's **Event()** function.

Note that the *LOOP_WINDOW* must be modal or it will not work properly. The do/while loop is waiting for either *LOOP_OK* or *LOOP_CANCEL* which are user-defined *EVENT_TYPE* values sent by **OK** and **Cancel** buttons. Because all other events are being sent to the *windowManager* this window must be modal, or other windows or objects could receive messages.

```
case CONTROL_LOOP:
```

```

{
LOOP_WINDOW *window = new LOOP_WINDOW("test.dat~LOOP_WINDOW");
>windowManager + window;

// Make sure the window is modal.
UI_EVENT event;
EVENT_TYPE ccode = 0;
do
{
eventManager->Get(event, Q_NORMAL);
ccode = event.type;

// LOOP_OK and LOOP_CANCEL are user defined values (>=10000)
// and programmed into the buttons that are on the LOOP_WINDOW.
if (ccode == LOOP_OK || ccode == LOOP_CANCEL)
>windowManager - window;
else
ccode = windowManager->Event(event);
} while (windowManager->Index(window) != -1);
>windowManager - window;
delete window;
}
break;

```

The control loop will break when one of two conditions occurs.

- Either *LOOP_OK* came through, or
- *LOOP_CANCEL*. If *LOOP_CANCEL* was seen the window is subtracted from the *windowManager* and the second part of the **while** loop test will fail and end the loop.

3.3 Interpretation of events in OpenZinc

ZD-TN4572

Keywords: event interpretation
 Versions: 3.0 and later
 Components: Library
 Platforms: All
 Issued: July 7, 1994

Question

How do events get interpreted in OpenZinc?

Answer

All raw events—for example, those coming from a keyboard or mouse—are passed to the receiving object. The receiving object's **Event()** function interprets and processes the event if it pertains to that object.

The following concepts are important:

One system-wide event map table contains entries for each object, and all logical events that correspond to a given raw event for that object. The event map table is a member variable of **UI_WINDOW_OBJECT**. The default event map table is located in `g_event.cpp`. The definition is:

```
UI_EVENT_MAP *eventMapTable;
```

The programmer may modify this table, or reassign the event map-table to a programmer defined table.

All objects contain an *eventMapTable* pointer since they all inherit from **UI_WINDOW_OBJECT**. When instantiated, each object points to the default (system-wide) *eventMapTable*.

All objects contain a virtual **Event()** member function. While most changes to event handling can be handled with changes to the default *eventMapTables*, the **Event()** function can be easily overloaded to handle programmer defined events, etc.

Example

For our example, a text field is the current object. Its **Event()** function demonstrates how OpenZinc processes and translates a raw event.

```
EVENT_TYPE UIW_TEXT::Event(const UI_EVENT &event)
{
    // get logical event from event map table
    EVENT_TYPE ccode = LogicalEvent(event, ID_TEXT);

    // process event
    switch (ccode)
    {
```

```

case S_CREATE:
case S_SIZE:

case OTHER_EVENTS:

// any unprocessed events are passed to base class for processing
default:
    ccode = UIW_WINDOW::Event(event);
    break;
}

```

Internal Details

LogicalEvent() searches the event map table, finds an exact match for the event, event raw code and object (e.g. window, button etc), or finds a match to the closest object from which the object is inherited and returns the logical event associated with this match, or if no match is found, no mapping occurs.

LogicalEvent() lives in **z_win2.cpp**, **MapEvent()** in **z_map.cpp**. To fully understand the algorithm, you must first understand *windowID*, which shows inheritance—the entries for *windowID[]*. For example, a **UIW_GROUP** would have:

```

windowID[0] = ID_GROUP;
windowID[1] = ID_WINDOW;
windowID[2] = ID_WINDOW_OBJECT
windowID[3] = ID_WINDOW_OBJECT
windowID[4] = ID_WINDOW_OBJECT

```

To modify the *eventMapTable* and reassign it to an object, refer to the following example:

```

#define MY_EVENT 10001
static UI_EVENT_MAP _eventMapTable[] = {
    { ID_TEXT, S_CLOSE, WM_KEYUP, F8 },
    { ID_TEXT, L_TOP, WM_KEYDOWN, F7 },
    // Define my new event in the event table
    { ID_TEXT, MY_EVENT, WM_CHAR, F6 },
    // End of array.
    { ID_END, 0, 0, 0 } };

// then in code, assign the new table to a text object ...
UIW_TEXT *text = new UIW_TEXT(2,2,20,20,"",50);
text->eventMapTable = _eventMapTable;

```

If this is done, when the text object is current, only the above logical events will be mapped by the text's **Event()** function.

Interpretation of events in OpenZinc ZD-TN4572

Notes:

- When a user clicks with the mouse on a text field, the raw mouse event code is placed in the event queue and the event is passed to the text object.
- The text object's `Event()` function calls `LogicalEvent()`, which in turn calls `MapEvent()`.
- The raw event is mapped to a logical event (by looking it up in an *eventMapTable*) and which is then returned by `LogicalEvent()`. In this case, the logical event will be `L_BEGIN_MARK`, and the `x/y` information in the event will indicate where in the text field to begin marking. (Alternatively, if a button were the current object, this same mouse click would generate an `LJSELECT` for the `ccode`, and the `Event()` function for the button would process it accordingly.)
- The `Event()` function acts on the logical event. (You might have the `Event()` function pass control to your callback function here, or send another event to a programmer-defined object.)

3.4 Trapping the S_CLOSE event

ZD-TN1010

Keywords: S_CLOSE
 Versions: 3.0+
 Components: UI_WINDOW_MANAGER.
 Platforms: All
 Issued: January 20, 1995

Question

Where can I trap *S_CLOSE*!

Answer

Here are two ways of trapping *S_CLOSE*.

The first way is to trap for *S_DEINITIALIZE* in the `Event()` function of your derived window. Once the `Event` function receives the event, you know that the window is about to be closed by the Window Manager. When this happens you can do any special cleanup or handling before passing the event to the base class. The drawback to this method is that the window is already in the process of being closed and shouldn't be blocked. The following snippet of code illustrates.

```
EVENT_TYPE DERIVED_WINDOW::Event(const UI_EVENT &event)
{

    case S_DEINITIALIZE: // Done in the Event() function of my derived window.
    {
        // Do any data handling or cleanup, then pass it to the base class.
        code = UIW_WINDOW::Event(event);
    }

    break;
```

The second way is deriving your own Window Manager. When *S_CLOSE* is put on the queue, the Window Manager will see it first, since it handles this type of message. Once it sees the event, you can continue to delete the window, or block the event upon conditions you define. The following snippet illustrates.

```
EVENT_TYPE NEW_WINDOW_MANAGER::Event(const UI_EVENT &event)
{

    case S_CLOSE: // Trap the S_CLOSE to swap the windows.
    {
        // Get a pointer to the old window.
        UI_WINDOW_OBJECT *currentWindow = First(); // current == first.
        // Make sure it is OK to delete the window.
```

Trapping the S_CLOSE event ZD-TN1010

```
if (deleteWin == OK)
    ccode = UI_WINDOW_MANAGER::Event(event);

// Else whatever condition is met to keep the window.
}
break;
```

3-5 Working with Flags

ZD-TN7000

Keywords: Flags
Versions: 3.0+
Platforms: All
Issued: January 20, 1995

Question

What are flags, and what do they do?

Answer

Flags are what you use to set and check information contained in objects; they provide more functionality to objects than just their class. For example, a UIW_BUTTON object uses flags to determine whether it is a radio button, a check box, or a button that appears three-dimensional.

Defined more technically, flags are const declarations, located in class definitions, with bits set to ones in binary places (2^0 , 2^1 , 2^2 , etc.). For readability by humans, the names of flags reflect what types of classes the flag affects. For example, the window object flag WOF_JUSTIFY_CENTER is

```
const WOF_JUSTIFY_CENTER = 00000001
```

the flag WOF_BORDER is

```
const WOF_BORDER = 00000100
```

and the flag WOF_MINICELL is

```
const WOF_MINICELL = 01000000
```

This way, a window object can be center justified and bordered, and use minicell coordinates, at the same time.

The following is a table of flag types, their naming conventions, and an example of how they affect the status or attributes of objects.

Flag naming convention

Flag type	Class	Example
Attachments	ATCF_FLAGS	ATCF_STRETCH
Buttons	BTF_FLAGS	BTF_RADIO_BUTTON
Button status	BTS_FLAGS	BTS_DEPRESSED
Dimension constraints	DNCF_FLAGS	DNCF_WIDTH
Dates	DTF_FLAGS	DTF_ASIAN_FORMAT
Icons	ICF_FLAGS	ICF_DOUBLE_CLICK
Images	IMFJFLAGS	IMF_HLED
Menu items	MNIF_FLAGS	MNIF_MINIMIZE
Numbers	NMF_FLAGS	NMF_SCIENTIFIC
Relative constraints	RLCF_FLAGS	RLCF_VERTICAL_CENTER
Strings	STF_FLAGS	STF_UPPER_CASE
System buttons	SYF_FLAGS	SYF_GENERIC
Storage	UIS_FLAGS	UIS_BINARY
Lists	WNF_FLAGS	WNF_AUTO_SORT
Window object actions	WOAF_FLAGS	WOAF_MDI_OBJECT
Window objects	WOF_FLAGS	WOF_NON_FIELD_REGION
Window object status	WOS_FLAGS	WOS_SELECTED

Flag functionality

Although there are flag types for nearly every window object in OpenZinc, flags do only one of two things: check status or set status. There are only a few status flags, and quite a few other flags. Use the status flags to determine if the user has changed your object in some way. Use the status flags to set or change the attributes of your objects.

- status flags
 - Flag prefixes that end in "S," as in WOS_. These reflect the status of an object, such as whether the user minimized or maximized it, or whether the user clicked in it.

- other flags
 - Flag prefixes that end in "F," as in WOAF_. These set the attributes of an object, such as whether the object takes up the entire window region, is lowercase, or is a vertical or horizontal scroll bar.

Where to find a complete list of flags

As we mentioned earlier, flags are really declarations of constants. To learn about the different flags that OpenZinc objects can use, you have several resources:

- The Quick Reference Guide
 - Lists the flags that each class uses, as well as the hexadecimal values of the flags.
- *Programmer's Reference* Vols. 1 and 2
 - Lists an class's different members, including the flag-setting member variables.
- The source code
 - You can always read the source code to find out the flags a class uses, and find out what happens to an object when it receives a specific flag after it's constructed.

How do I use flags?

You use flags to set or check the state of an object. You can use them three ways.

1. You can use an object's flag-checking functions.
2. You can set the flags in the object's constructor when you use the new operator.

3. After the object's construction, you can

- set a flag into an object's flag member with a bitwise OR
- unset a flag into an object's flag member with a bitwise AND
- toggle a flag into an object's flag member with a bitwise XOR.

Using an object's flag-checking functions

Each object class definition contains some flag-checking functions that you can use with an object flag member. The functions return TRUE if the condition is met, or FALSE if it is not.

An example of a flag-checking function is UI_WINDOW_OBJECT's IsLeftJustified(). Here's an example of how you can use it:

```
UIW_STRING *myString;

    if (myString->IsLeftJustified( ))
        \\ do something
```

There are quite a few flag-checking functions you can use. For an exhaustive list of them, consult Table 2, "Flag-checking functions," on page 42.

Working with Flags ZD-TN7000

Setting flags in the object's constructor with a new operator

To set a flag in the object's constructor with the new operator, do the following:

```
UIW_TEXT *myTextObject = new UIW_TEXT(0, 0, 60, 20, "", 256, WNF_NO_FLAGS, WOF_NON_SELECTABLE);
```

Setting flags with the flag members

To set a flag with an object's woFlags member, use a bitwise OR. If the object is attached to the Window Manager, call the object's Information() function to cause it to redisplay itself.

```
object->woFlags |= WOF_NON_SELECTABLE; \\ set the object as nonselectable
object->Information(I_CHANGED_FLAGS, ZIL_NULLP(void); \\ tell the object its flag changed
only if attached to Window Manager
```

To unset a flag with an object's woFlags member, use a bitwise AND. If the object is attached to the Window Manager, call the object's Information() function to cause the object to redisplay itself.

```
object->woFlags &= ~WOF_NON_SELECTABLE; \\ set the object as nonselectable
object->Information(I_CHANGED_FLAGS, ZIL_NULLP(void); \\ tell the object its flag changed
only if attached to Window Manager
```

To toggle a flag with an object's woFlags member, use a bitwise XOR. If the object is attached to the Window Manager, call the object's Information() function to cause the object to redisplay itself.

```
object->woFlags ^= WOF_NON_SELECTABLE; \\ set the object as nonselectable
object->Information(I_CHANGED_FLAGS, ZIL_NULLP(void); \\ tell the object its flag changed
only if attached to Window Manager
```

Section 4

Drawing, video, and graphics

The DrawItem() function ZD-TN1007

1.0 The DrawItem() function ZD-TN1007

Keywords: DrawItem(), drawing, VirtualGet(), VirtualPut()
Versions: All
Components: DrawItem()
Platforms: All
Issued: November 11, 1994

Question

How do I use the DrawItem() function?

Answer

Use the DrawItem() function when you want an object to appear differently than its default appearance, or when you want additional graphics drawn on a generic object. The generic format follows.

```
EVENT_TYPE CLASS_NAME::DrawItem(const UI_EVENT &event, EVENT_TYPE ccode)
{
    UI_REGION region=trueRegion;
    display->VirtualGet(screenID, region);
    // Possibly call the base class (if you'll draw on "top" of the standard object)
    BASE_CLASS::DrawItem(event, ccode);

    // Programmer defined drawing (use display class primitives)

    display->VirtualPut(screenID);

    return(TRUE);
}
```

If the object is derived from one of OpenZinc's class objects then the user may want to call the base class DrawItem() function before he does his drawing (which might draw on top of the default object.) At the end of the DrawItem() function we return *TRUE* to indicate that the drawing is complete and no more drawing should take place. If *FALSE* or 0 is returned then additional drawing may take place by the base class. VirtualGet() and VirtualPut() functions attempt to optimize the amount of drawing done to the screen.

Drawing should be based on the true coordinates and not relative; "trueRegion" is a *UI_REGION* structure and has the proper region assigned to it. If the programmer draws outside of this region then there may be unexpected results. If drawing could occur outside of this region, such as a long line, then an additional member, clip, also a *UI_REGION* structure, should be passed to the drawing primitives. This will ensure that any drawing outside of "trueRegion" will be properly clipped; "trueRegion" is a public member and "clip" a protected member of *UI_WINDOW_OBJECT*.

WOS_OWNERDRAW must be set in the object's *woStatus* member for this function to be called. For additional information consult the Getting Started manual under the LSTITM example.

4.2 Multithreaded applications on OS/2

ZD-TN1017

Keywords: Event manager.
 Versions: 4.0+
 Components: UI_EVENT_MANAGER
 Platforms: OS/2

Question

How do I get OpenZinc to work with multiple threads?

Answer

There are several things that must be done to get OpenZinc to function with multiple threads. OpenZinc is a non-reentrant application. What this means is that a call to a function can't be interrupted by a subsequent call to the same function or the variables will become corrupted. Most C library functions are this way, and so is OpenZinc.

To get around this you must derive the Event Manager and override the Put() function. Since this is a virtual function it very simple to do. Here is a typical class declaration that you might use with the Event Manager.

```
class ZIL_EXPORT_CLASS EVENT_MANAGER : public UI_EVENT_MANAGER
{
public:
    EVENT_MANAGER(UI_DISPLAY *_display, int _noOfElements = 100);
    virtual ~EVENT_MANAGER();
    void Put(const UI_EVENT &event, Q_FLAGS flags);
};
```

Since the Put() function is non-reentrant it must be protected or the event queue will become corrupted. There are two ways to do this. You can either use the OS/2 functions, DosEnterCritSec() and DosExitCritSec() functions which is very simple or you can use the functions DosCreateMutexSem() and DosCloseMutexSem().

The Enter and Exit critical section functions take no arguments and are therefore simple to use. The example uses this method as illustrated by the following snippet of code.

```
void EVENT_MANAGER::Put(const UI_EVENT &event, Q_FLAGS flags)
{
    // Start the critical section.
    DosEnterCritSec();
    UI_EVENT_MANAGER::Put(event, flags);
    // Exit the critical section.
    DosExitCritSec();
}
```

Multithreaded applications on OS/2 ZD-TN1017

True OS/2 gurus would probably argue that this should be done with semaphores, using the other mentioned functions. You should consult your OS/2 programming manual on the use of the other functions. They create and delete mutual exclusive semaphores.

Once you have protected the Put() function this allows your other threads to post messages on the event queue giving you the ability to communicate with OpenZinc. The only constraint on this is that OpenZinc must remain on a single thread. It can't at this time be split apart into multiple threads. Threading allows you to run other programs or processes next to your OpenZinc program.

4.3 Changing video modes at run time

ZD-TN2005

Key words: display mode, resolution, run time

Versions: OpenZinc 3.5 and later

Components: Library

Platforms: DOS

Issued: September 15, 1994

Question

How do you change graphics or text resolution at run time?

Answer

Changing the display mode requires:

- Notifying the Window Manager and Event Manager that the display is being changed;
- Deleting the current display;
- Creating the new display; and
- Notifying the Window Manager and Event Manager of the new display.

The following is an example program that can change between two possible text modes and two possible graphics modes. The graphics display used in this example is the **UI_GRAPHICS_DISPLAY**. The major portion of the **MY_WIN::Event()** function came from the **ZINCAPP.CPP** module in the **ZINCAPP** tutorial.

Note: If using OpenZinc 3.6 or OpenZinc 3.5 you will need to replace **ZIL_NULLF(ZIL_USER_FUNCTION)** with **NULLF** and **ZIL_NULLP(void)** with **NULLP(void)**.

```
#include <ui_win.hpp>

const int TDM_25X40 = 10000;
const int TDM_43X80 = 10001;
const int GR_DEFAULT = 10002;
const int GR_102    = 10003;

class MY_WIN : public UIW_WINDOW

public:
    MY_WIN();
    virtual ~MY_WIN(){}
    virtual EVENT_TYPE Event(const UI_EVENT &event);
};

MY_WIN :MY_WIN() : UIW_WINDOW(0, 0, 50, 10)
```

Changing video modes at run time ZD-TN2005

```
{
    *this
    + new UIW_BORDER
    + new UIW_MAXIMIZE_BUTTON
    + new UIW_MINIMIZE_BUTTON
    + new UIW_SYSTEM_BUTTON(SYF_GENERIC)
    + new UIW_TITLE("TEST")
    + &(*new UIW_PULL_DOWN_MENU()
        + &(*new UIW_PULL_DOWN_ITEM("Text Mode")
            + new UIW_POP_UP_ITEM("TDM_25", MNIF_SEND_MESSAGE, BTF_N0_3D,
                WOF_NO_FLAGS, ZIL_NULLF(ZIL_USER_FUNCTION), TDM_25X40)
            + new UIW_POP_UP_ITEM("TDM_43", MNIF_SEND_MESSAGE, BTF_N0_3D,
                WGF_NO_FLAGS, ZIL_NULLF(ZIL_USER_FUNCTION), TDM_43X80))
        + &(*new UIW_PULL_DOWN_ITEM("Graphics Mode")
            + new UIW_POP_UP_ITEM("Default", MNIF_SEND_MESSAGE, BTF_N0_3D,
                WOF_NO_FLAGS, ZIL_NULLF(ZIL_USER_FUNCTION), GR_DEFAULT)
            + new UIW_POP_UP_ITEM("SuperVGA", MNIF_SEND_MESSAGE, BTF_N0_3D,
                WOF_NO_FLAGS, ZIL_NULLF(ZIL_USER_FUNCTION), GR_102)));

EVENT_TYPE MY_WIN::Event(const UI_EVENT &event)
{
    UI_EVENT tEvent;
    int isGraphics = FALSE;
    EVENT_TYPE ccode = LogicalEvent(event);

    // Check if the event is a OpenZinc system event,
    if (ccode < TDM_25X40)
        return (UIW_WINDOW::Event(event));
    else
    {
        switch(ccode)
        {
        case TDM_25X40:
            tEvent.rawCode = TDM_25x40;
            break;
        case TDM_43X80:
            tEvent.rawCode = TDM_43x80;
            break;
        case GR_DEFAULT:
            tEvent.rawCode = 4;
            isGraphics = TRUE;
            break;
        case GR_102:
            tEvent.rawCode = 0x102;
            isGraphics = TRUE;
            break;
        }
        tEvent.type = (S_RESET_DISPLAY);
    }
}
```

```

    tEvent.data = ZIL_NULLP(void);

// Notify the event and window managers that we are changing the display.
windowManager->Event(tEvent);
eventManager->Event(tEvent);
delete display;

// Check for text or graphics mode.
if (isGraphics)
{
    display = new UI_GRAPHICS_DISPLAY(tEvent.rawCode);
    if (!display->installed)
    {
        delete display;
        display = new UI_TEXT_DISPLAY();
    }
}
else
    display = new UI_TEXT_DISPLAY(tEvent.rawCode);

// Make sure the window doesn't have negative coordinates. If you have
// more than one window, this will need to be done for each window.
if (relative.top < 0)
{
    relative.bottom = relative.Height() - 1;
    relative.top = 0;
}
if (relative.left < 0)
{
    relative.right = relative.Width() - 1;
    relative.left = 0;
}

// Notify the event and window managers that we changed the display.
tEvent.data = display;
eventManager->Event(tEvent);
ccode = windowManager->Event(tEvent);
}
return(ccode);
}

int UI_APPLICATION::Main (void)
{
    UIW_WINDOW *window = new MY_WIN();
    *windowManager
    + window;
    UI_APPLICATION::Control();
}

```


Changing video modes at run time ZD-TN2005

```
// Must re-initialize the UI_APPLICATION class' display member to exit gracefully,  
display = UI_WINDOW_OBJECT::display ;  
return(0);  
}
```

4.4	Palette mapping in OpenZinc	ZD-TN4573
Keywords:	color, palette mapping	
Versions:	3.0 and later	
Components:	Library	
Platforms:	All	
Issued:	July 7, 1994	
File:	PALETTE.ZIP	

Question

How are object colors handled in OpenZinc?

Answer

Palette mapping (color control) under OpenZinc follows these rules:

- The native operating system will draw objects in the native OS colors, unless the *WOSjOWNERDRAW* flag is set for that object.
- In DOS and Curses OpenZinc's **DrawItem()** function will be called to draw each object.
- If the *WOS_OWNERDRAW* flag is set, the following applies:
 - In DOS, OpenZinc will always call a programmer-defined **DrawItem()** function for the drawing of the object.
 - In other operating systems, the OS will defer the drawing of that object to OpenZinc's **DrawItem()** function for the object or to a programmer defined **DrawItem()** function, if defined.
- Each object has a pointer to a palette map table which contains entries for each object, possible states for that object (e.g. selected, current, noncurrent etc..) and a corresponding set of colors for that object's state.
- This table can be directly modified by the programmer or the programmer can reassign an object's palette map table to a programmer defined table.
- Objects are drawn (or redrawn) whenever the object or its parent receives a message to draw. For example, objects are redrawn when they receive *SJREDISPLAY* or *S_SIZE* messages, etc.
- The palette map table is a member of **UI_WINDOW_OBJECT** and is defined as:

```
UI_PALETTE_MAP *paletteMapTable;
```

Palette mapping in OpenZinc ZD-TN4573

- All objects contain a *paletteMapTable* pointer since they all inherit from **UI_WINDOW_OBJECT**. Upon instantiation, each object's *paletteMapTable* pointer will point to the default palette map table found in **g_pnorm.cpp**.

Example

```
EVENT_TYPE UIW_STRING::DrawItem(const UI_EVENT &, EVENT_TYPE ccode)
// The DrawItem() function is defined virtual so the base class's DrawItem() function //
can be overridden by a programmer defined DrawItem() function.
{
    UI_REGION region=trueRegion;

    // lastPalette is set under the S_DISPLAY_ACTIVE / S_DISPLAY_INACTIVE
    // event in the string's Event() function by a call to the LogicalPalette()
    // function in z_map1.cpp. It calls MapPalette() in z_win2.cpp.
    display->Rectangle(screenID, region, lastPalette, 0, TRUE, FALSE, &clip);

    DrawText(screenID, region, text, lastPalette, FALSE, ccode);

    return (ccode);
}
```

To fully understand this algorithm, the following concepts are important:

windowID[] shows inheritance. The entries for *windowID[]* for a **UIW_GROUP** object would be

```
windowID[0] = ID_GROUP;
windowID[1] = ID_WINDOW;
windowID[2] = ID_WINDOW_OBJECT;
windowID[3] = ID_WINDOW_OBJECT;
windowID[4] = ID_WINDOW_OBJECT;
```

FlagSet() and **FlagsSet()** are defined as follows:

```
#define FlagSet(flag1, flag2) ((flag1) & (flag2))
#define FlagsSet(flag1, flag2) (((flag1) & (flag2)) == (flag2))
```

LOGICAL_PALETTE values:

```
const LOGICAL_PALETTE PM_ANY= 0x0000;
const LOGICAL_PALETTE PM_ACTIVE= 0x0001;
const LOGICAL_PALETTE PM_INACTIVE= 0x0002;
const LOGICAL_PALETTE PM_CURRENT= 0x0004;
const LOGICAL_PALETTE PM_SELECTED= 0x0008;
const LOGICAL_PALETTE PM_NON_SELECTABLE= 0x0010;
const LOGICAL_PALETTE PM_HOT_KEY= 0x0020;
// Special mode palettes (WOAF_DIALOG_OBJECT border)
const LOGICAL_PALETTE PM_SPECIAL= 0x0040;
```

- An analysis of the **LogicalPalette()** and **MapPalette()** algorithms shows an important concept for palette map table entries: *PM_ANY* must be at the end of an object's grouping of *UI_PALETTE_MAP* states.
- To create a programmer-defined palette mapping for an object create a *UI_PALETTE_MAP* for the object, add the specific entries, then assign *object->paletteMapTable* this new *paletteMap*.

```

static UI_PALETTE_MAP newPaletteMap[] =
{
    { ID_WINDOW_OBJECT, PM_CURRENT,
    { '  attrib(YELLOW, BLACK), attrib(MONO_HIGH, MONO_BLACK), PTN_SOLID_FILL,
    BLUE, WHITE, BW_BLACK, BW_WHITE, GSBLACK, GS_WHITE } },
    { ID_WINDOW_OBJECT, PM_ANY,
    { '  attrib(YELLOW, LIGHTGRAY), attrib(MONO_HIGH, MONO_BLACK),
    PTN_SOLID_FILL, WHITE, BLUE, BW_BLACK, BW_WHITE, GS_BLACK, GS_WHITE } },
    // End of array.
    { ID_END, 0, { 0, 0, 0, 0, 0, 0 } }
};
object->paletteMapTable = newPaletteMap;

```

Loading large bitmaps with OS/2-specific functions ZD-TN1011

4.5 Loading large bitmaps with OS/2-specific functions ZD-TN1011

Keywords: bitmaps, OS/2
Versions: 4.0+
Components: UI_WINDOW_OBJECT
Platforms: OS/2

Question

How do I display bitmaps larger than 128x128 under OS/2?

Answer

A bitmap ID is assigned in a resource (.RC) file that corresponds to an OS/2 .BMP file. With that ID, a bitmap can be loaded using OS/2 calls. Once the bitmap is loaded it can be easily displayed using OpenZinc's UI_WINDOW_OBJECT class. The OS/2 function GpiLoadBitmap takes the bitmapID, loads the bitmap and returns a handle to the bitmap. It is also necessary to get a handle to the presentation space with WinGetPS(). Once that is done then the call to WinDrawBitmap() can be used to display the bitmap as the following code shows.

```
BITMAP_OBJECT::BITMAP_OBJECT(int _bitmapID) :
    UI_WINDOW_OBJECT(0, 0, 0, 0, WOF_NON_FIELD_REGION, WOAF_NON_CURRENT),
    bitmapID(JoitmapID), hBitmap(0)
{

BITMAP_OBJECT::~~BITMAP_OBJECT()
{
    //We need to make sure that we delete the bitmap when we destroy the object.
    If (hBitmap)
        GpiDeleteBitmap(hBitmap);
}

// Displaying the bitmap is done here.
EVENT_TYPE BITMAP_OBJECT::DrawItem(const UI_EVENT &, EVENT_TYPE)
{
    UI_REGION region=trueRegion;
    display->VirtualGet(screenID, region);

    //We create an RECTL structure and fill it with our objects region to prepare for the
    drawing of the bitmap.
    RECTL rect;
    rect.xLeft = 0;
    rect.yBottom = 0;
    rect.xRight=trueRegion.WidthQ-1;
    rect.yTop=trueRegion.Height()-1;
```

Loading large bitmaps with OS/2-specific functions ZD-TN1011

```
// The bitmap is drawn with WinDrawBitmap.
WinDrawBitmap(hps, hBitmap, NULL, (POINTL *)&rect, 0, 0,
    DBM_IMAGEATTRS I
    (FlagSet(woFlags, WOF_NON_FIELD_REGION) ? DBM_STRETCH : 0));
display->VirtualPut(screenID);
return(TRUE);
}
EVENT_TYPE BITMAP_OBJECT::Event(const UI_EVENT &event)
{
    EVENT_TYPE ccode = event.type;
    switch (ccode)
    {
        case S_CREATE:
            // Since we are doing the loading and displaying of the bitmap the object must be marked
            // with
            // WOS_OWNERDRAW flag.
            woStatus |= WOS_OWNERDRAW;
            //In the event function under S_CREATE we first get a handle to the presentation space //
            // by calling WinGetPS then make the call to GpiLoadBitmap with the returned
            // presentation space handle.
            ccode = UI_WINDOW_OBJECT::Event(event);
            hps = WinGetPS(screenID);
            hBitmap = GpiLoadBitmap(hps, NULLHANDLE, bitmapID, 0, 0);
            break;
        default:
            ccode = UI_WINDOW_OBJECT::Event(event);
    }
    return ccode;
}
```

A

4.6 Setting the mouse cursor to a wait state

ZD-TN4020

Keywords: mouse, changing to an hourglass.
 Version: 3.6,4.0
 Component: Mouse; eventManager
 Platforms: Windows
 Issued: January 31, 1995

Question

How can I change the mouse cursor image an hourglass so that when the user moves the mouse, it doesn't change back to a normal view cursor mouse image?

Answer

In Windows, each object determines how the mouse cursor image is to be displayed when the mouse is moved over that object. This means that when the mouse gets changed to an hourglass image (*DM_WAIT*), as soon as the user moves the mouse, the object that the mouse is over gets a mouse move event, and changes the mouse cursor image back to the image that object wants.

One simple way to force the mouse to remain as an hourglass, even when the user moves the mouse, is to trap all mouse events before they get passed to the *windowManager*, and not pass any mouse events to the *windowManager* that would change the mouse image (such as *WM_MOUSEMOVE*). This can be done writing your own **Control()** loop as follows:

```
// Use the following code in place of UI_APPLICATION::Control();
UI_EVENT event;
EVENT_TYPE ccode;
do
{
    // get an event from the queue
    eventManager->Get(event, Q_NORMAL);

    // pass mouse event to windowManager only when not in hourglass state.
    // When in hourglass state, windowManger->userFlags gets set by programmer
    // to 1.
    if (!(windowManager->userFlags)
        && event.message.message ==WM_MOUSEMOVE)
        ccode = windowManager->Event(event);
    else

        eventManager->DeviceImage(E_MOUSE,DM_WAIT);

} while (ccode != L_EXIT && ccode != S_NO_OBJECT);
```

In order for this to work, the mouse would have to be changed to an hourglass with the following line of code:

```
eventManager->DeviceImage(E_MOUSE, DM_WAIT);
```

Setting the mouse cursor to a wait state ZD-TN4020

Then `windowManager->userFlags` could be set to a pointer to some object of window. This would cause the `if()` statement to fail and no mouse move events would get processed which would force the mouse to remain as an hourglass image.

When you want the hourglass image to be changed back, use the following:

```
eventManager->DeviceImage(E_MOUSE, DM_VIEW);  
windowManager->userFlags =0;
```


Setting the mouse cursor to a wait state ZD-TN4020

Section 5

OpenZinc Designer

5.1 Integrating Directory Services from OpenZinc Designer ZD-TN1009

Keywords: Services, OpenZinc Designer
 Versions: 4.0+
 Components: Directory services from OpenZinc Designer.
 Platforms: All
 Issued: November 21, 1994

Question

How do I use the directory services incorporated in OpenZinc Designer?

Answer

In versions of OpenZinc previous to version 4.0, directory services have existed only as an example program called **FILEEDIT**. With version 4.0 this example has been eliminated, and the directory services directly supported as a component of OpenZinc Designer. This means that directory services has been added to the Designer libraries, and, even better, has been implemented as a separate module so the user can integrate this into his own code.

To make directory services available to your code:

In your code file you need to include **DIRECT.HPP** and in your makefile or project you need to include **SERVICE.LIB** and **DIRECT.LIB**.

The header should be in your `\OpenZinc\INCLUDE` directory and the **.LIB** files should be in the proper library directory. If these files are not present, you can remake the Designer or just compile the libraries in `\OpenZinc\DESIGNSERVICE` and `\OpenZinc\DESIGNDIRECT`.

To access directory services in your application you must create a new service manager and a new directory services:

```
// Create service manager to handle storage for various services
_serviceManager = new ZAF_SERVICE_MANAGER;
// Create directory service
*_serviceManager + new ZAF_DIRECTORY_SERVICE;
```

Various services take requests and return answers to those requests. You first need to set up the requestor, the request, then you activate the service as follows:

```
int myRequest = OPT_FILE_OPEN;
// Set up the requestor
directoryService->Information(I_SET_REQUESTOR, this);
// Make a request to the directory service.
directoryService->Information(I_SET_REQUEST, &myRequest);
// Activate the service.
directoryService->Information(X_ACTIVATE_SERVICE, ZIL_NULLP(void));
```

Integrating Directory Services from OpenZinc Designer ZD-TN1009

The requestor is the object which will receive the return event. The request is what you want the service to do. In this case the requestor is set as the derived window and the request to open a file. When terminated, the service will return the negative value of the original request (when you choose OK on the service window.) The returned request is put into the data portion of the event that is passed back to the window.

The following is a simple program that shows how to implement the directory service. You should look closely at the **Event()** function of the derived window. This program assumes that there is a resource with a menu or button with the value 10000 assigned to it, and that there is a **UIW_STRING** field with the *stringID* of **STRING_FIELD**. The program takes the return value and copies the data into the string field.

```
#include <ui_win.hpp>
#include <direct.hpp>

const EVENT_TYPE OPEN_FILE = 10000;

class WINDOW : public UIW_WINDOW
{
public:
    WINDOW();
    virtual ~WINDOW() {}
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    ZAF_DIRECTORY_SERVICE *directoryService;

protected:
    UI_WINDOW_OBJECT *field;
};

WINDOW::WINDOW() : UIW_WINDOW("p_main.dat~MAIN_WINDOW")
{
    directoryService = new ZAF_DIRECTORY_SERVICE;
    field = Get("STRING_FIELD");
    windowManager->Center(this);
}

EVENT_TYPE WINDOW::Event(const UI_EVENT &event)
{
    EVENT_TYPE ccode = LogicalEvent(event);
    switch(ccode)
    {
        case OPEN_FILE:
            EVENT_TYPE myRequest = OPT_FILE_OPEN;
            // Set the requestor to be this window.
            directoryService->Information(I_SET_REQUESTOR, this);
            // Make a request to the directory service.
            directoryService->Information(I_SET_REQUEST, ScirtyRequest);
            // Activate the service.
            directoryService->Information(I_ACTIVATE_SERVICE,ZIL_NULLP(void));
    }
}
```

```
    }
    break;

    case -OPT_FILE_OPEN: // The return request is the opposite of the initial request.
        field->Information(I_SET_TEXT, event.data);
        break;

    default:
        ccode = UIW_WINDOW::Event(event);
    }
    return(ccode);
}

UI_APPLICATION::Main()
{
    WINDOW *win = new WINDOW;

    _serviceManager = new ZAF_SERVICE_MANAGER;
    *_serviceManager + win->directoryService;

    *windovManager + win;

    Control();

    delete _serviceManager;
    return(0);
}
```

5.2 The Service Manager

ZD-TN1015

Keywords: Service Manager
Versions: 4.0
Components: ZAF_SERVICE_MANAGER
Platforms: All

Question

Why do I need the Service Manager to use the directory service?

Answer

By default the Service Manager overrides the error system, help system, and the exit function. Because of this, you should override the Service Manager's allocation of these systems when using your own.

Service Manager manages the storage files for the services. The services request files from the Service Manager, then the Service Manager opens the file. After opening the file, the Service Manager returns the storage file pointer to the requesting service.

The Service Manager works similarly to the Window Manager. While the Window Manager manages the message passing to attached windows, the Service Manager handles attached services. This makes services reusable through the life of the program without instantiating the services over and over.

To override the Service Manager's errorSystem, helpSystem, and ExitFunction, assign them after allocating the Service Manager itself. Make sure that the memory to errorSystem and helpSystem is deleted before you reassign them, because the Service Manager has already allocated them. If you reallocate them without deleting them first, your program will have a memory leak.

```
// The Service Manager allocates the helpSystem and errorSystem.
_serviceManager = new ZAF_SERVICE_MANAGER;
//If you want your copy of these they must be set after allocating the Service Manager.
// Delete the previous copy or you will have a leak,
delete UI_WINDOW_OBJECT::errorSystem;
UI_WINDOW_OBJECT::errorSystem = new UI_ERROR_SYSTEM;

// Reassign the exitFunction if needed. No Deletion is necessary for this.
windowManager->exitFunction = MyExitFunction;

// Process user responses.
Control();
// Clean up.
delete _serviceManager;
// Don't delete errorSystem or helpSystem, Service Manager is already removing them!
return (0);
```

5.3 Exporting a window to text

ZD-TN1016

Keywords: Export, .DAT, .TXT
 Versions: 4.0+
 Components: UIW.WINDOW
 Platforms: All

Question

What does exporting a .DAT file to a .TXT file give me?

Answer

Exporting a .DAT file to a .TXT file outlines the relationships and contents of the objects in the .DAT file in a text file. The contents of the text file are the object's stringID and text.

Exporting a .DAT file to a .TXT file outputs the stringID and the text but does not output the type of object. This means that exporting a .DAT file to text is a one way operation; you can't import a .TXT file back into the Designer. You could, however, enter the text into the Designer through delta storage, if your application must support multiple languages. You could also create separate .DAT files for each language. Exporting a .DAT file to a text is useful in translating. You can send this file to your translators who can then return you a translation of the text in your main application's GUI. You can then enter the translation into your .DAT file.

Here is an output of a simple window. This window has a menu with several popup items, a status bar, and a notebook with three pages and a few objects on each page.

```
WINDOW = "Window"
{
  FIELD_1
  {
    FIELD_2 = "item"
    {
      FIELD_44 = "item"
      FIELD_45 = "item"
      FIELD_46 = "item"
    }
  }
  FIELD_3
  {
    STATUS_STRING = "empty string"
    STATUS_PROMPT = "Status:"
  }
  NOTEBOOK
  {
    PAGE_1 = "Page 1"
    {
      FIELD_9 = "string"
    }
  }
}
```

Exporting a window to text ZD-TN1016

```
FIELD_10 = "string"
FIELD_11 = "string"
FIELD_12 = "button"
FIELD_13 = "button"
VERTICAL_LIST
{
    FIELD_15 = "Check-box"
    FIELD_16 = "Check-box"
    FIELD_17 = "Check-box"
    FIELD_18 = "Check-box"
    FIELD_19 = "Check-box"
}
}
PAGE_2 = "Page 2"
{
    HORIZONTAL_LIST
    {
        FIELD_21 = "Radio-button"
        FIELD_22 = "Radio-button"
        FIELD_23 = "Radio-button"
        FIELD_24 = "Radio-button"
        FIELD_25 = "Radio-button"
        FIELD_26 = "Radio-button"
        FIELD_27 = "Radio-button"
        FIELD_28 = "Radio-button"
        FIELD_29 = "Radio-button"
    }
    PHONE_FIELD = "(____)____-_____"
    DATE_FIELD = "5-2-1995"
    TIME_FIELD = "8:21 a.m."
}
PAGE_3 = "Page 3"
{
    MAIN_GROUP = "Main Group"
    {
        FIELD_34 = "Radio-button"
        FIELD_35 = "Radio-button"
        FIELD_36 = "Radio-button"
        FIELD_37 = "Radio-button"
    }
    HUE
    SATURATION
    VALUE
    FIELD_41 = "Hue:"
    FIELD_42 = "Sat:"
    FIELD_43 = "Val:"
}
}
}
```

5.4 Using multiple .DAT files in one application ZD-TN2003

Keywords: .DAT
 Versions: 3.5, 3.6, 4.0
 Components: .DAT files
 Platforms: All
 Issued: September 8, 1994

Question

Can I have more than one .DAT file in an application?

Answer

Yes, you can have more than one .DAT file in an application. However, there are some things that must be done to do this. The designer generates three files when saving a resource file: the .DAT file itself, a .CPP and an .HPP. The .CPP contains a user table and an object table. The .HPP file contains the help context and the *numberIDs* for the objects.

To use multiple .DAT files, the user tables and object tables from all of the .CPP files must be combined into one user table and one object table. When combining the object tables, there must only be one entry for each type of object. When combining the user tables, there must only be one entry for each user function or compare function. The .HPP files must also be included in the one .CPP file.

Following is a simple example of combining two .CPP files created by OpenZinc Designer 4.0. Both original .CPP files are listed first, followed by a step-by-step merge of the two files:

TEST1.CPP

```
#include <ui_win.hpp>
#define USE_DERIVED_OBJECTS
#include "test1.hpp"
void z_jump_dummy(void) { } // Bug fix for broken linkers.
extern EVENT_TYPE Function1(UI_WINDOW_OBJECT *, UI_EVENT &, EVENT_TYPE);
static ZIL_ICHAR _Function1[] = { 'F','u\n','c','t','i','o'. 'n','l',0 };

static UI_ITEM _userTable[] =
{
    { 0, ZIL_VOIDF(Function1), _Function1, 0 },
    { ID_END, ZIL_NULLP(void), ZIL_NULLP(ZIL_ICHAR), 0 }
};
UI_ITEM *UI_WINDOW_OBJECT::userTable = _userTable;

static UI_ITEM _objectTable[] =
{
    { ID_BORDER, ZIL_VOIDF(UIW_BORDER::New), UIW_BORDER::_className, 0 },
    { ID_BUTTON, ZIL_VOIDF(UIW_BUTTON::New), UIW_BUTTON::_className, 0 },
};
```


Using multiple .DAT files in one application ZD-TN2003

```
{ ID_MAXIMIZE_BUTTON, ZIL_VOIDF(UIW_MAXIMIZE_BUTTON::New),
  UIW_MAXIMIZE_BUTTON::_className, 0 },
{ ID_MINIMIZE_BUTTON, ZIL_VOIDF(UIW_MINIMIZE_BUTTON::New),
  UIW_MINIMIZE_BUTTON::_className, 0 },
{ ID_STRING, ZIL_VOIDF(UIW_STRING::New), UIW_STRING::_className, 0 },
{ ID_SYSTEM_BUTTON, ZIL_VOIDF(UIW_SYSTEM_BUTTON::New),
  UIW_SYSTEM_BUTTON::_className, 0 },
{ IDJTITLE, ZIL_VOIDF(UIW_TITLE::New), UIW_TITLE::_className, 0 },
{ IDJWINDOW, ZIL_VOIDF(UIW_WINDOW::New), UIW_WINDOW::_className, 0 },
{ ID_END, ZIL_NULLP(void), ZIL_NULLP(ZIL_ICHAR), 0 }
};
UI_ITEM *UI_WINDOW_OBJECT::objectTable = _objectTable;
```

TEST2.CPP

```
#include <ui_win.hpp>
#define USE_DERIVED_OBJECTS
#include "test2.hpp"
void zjunp_duinmy(void) { } // Bug fix for broken linkers,
extern EVENT_TYPE Func2(UI_WINDOW_OBJECT *, UI_EVENT &, EVENT_TYPE);
static ZIL_ICHAR _Func2[] = { 'F','u','n','c','2',0 };

static UI_ITEM _userTable[] =
{
  { 0, ZIL_VOIDF(Func2), _Func2, 0 },
  { ID_END, ZIL_NULLP(void), ZIL_NULLP(ZIL_ICHAR), 0 }
};
UI_ITEM *UI_WINDOW_OBJECT::userTable = _userTable;

static UI_ITEM _objectTable[] =
{
  { ID_BORDER, ZIL_VOIDF(UIW_BORDER::New), UIW_BORDER::_className, 0 },
  { ID_BUTTON, ZIL_VOIDF(UIW_BUTTON::New), UIW_BUTTON::_className, 0 },
  { ID_ICON, ZIL_VOIDF(UIW_ICON::New), UIW_ICON::_className, 0 },
  { ID_MAXIMIZE_BUTTON, ZIL_VOIDF(UIW_MAXIMIZE_BUTTON::New),
    UIW_MAXIMIZE_BUTTON::_className, 0 },
  { ID_MINIMIZE_BUTTON, ZIL_VOIDF(UIW_MINIMIZE_BUTTON::New),
    UIW_MINIMIZE_BUTTON::_className, 0 },
  { ID_SYSTEM_BUTTON, ZIL_VOIDF(UIW_SYSTEM_BUTTON::New),
    UIW_SYSTEM_BUTTON::_className, 0 },
  { ID_TITLE, ZIL_VOIDF(UIW_TITLE::New), UIW_TITLE::_className, 0 },
  { ID_WINDOW, ZIL_VOIDF(UIW_WINDOW::New), UIW_WINDOW::_className, 0 },
  { ID_END, ZIL_NULLP(void), ZIL_NULLP(ZIL_ICHAR), 0 }
};
UI_ITEM *UI_WINDOW_OBJECT::objectTable = _objectTable;
```

TEST1.CPP and TEST2.CPP combined

```
#include <ui_win.hpp>
#define USE_DERIVED_OBJECTS
#include "test1.hpp"
```

```

Must add the include for test2.hpp.
#include "test2.hpp"
void z_jump_durnrny(void) { } // Bug fix for broken linkers.
extern EVENT_TYPE Function1(UI_WINDOW_OBJECT *, UI_EVENT &, EVENT_TYPE);
Must add the function prototype.
extern EVENT_TYPE Func2(UI_WINDOW_OBJECT *, UI_EVENT &, EVENT_TYPE);
static ZIL_ICHAR_Function1t] = { 'F','u','n','c','1','o','n','1',0 };
Must add the international string for the function name,
static ZIL_ICHAR_Func2[] = { 'F','u','n','c','2',0 };

static UI_ITEM _userTable[] =
{
    { 0, ZIL_VOIDF(Function1), _Function1, 0 },
Must add the function to the user table.
    { 0, ZIL_VOIDF(Func2), _Func2, 0 },
    { ID_END, ZIL_NULLP(void), ZIL_NULLP(ZIL_ICHAR), 0 }
};
UI_ITEM *UI_WINDOW_OBJECT::userTable = _userTable;

static UI_ITEM _objectTable[] =
{
    { ID_BORDER, ZIL_VOIDF(UIW_BORDER::New), UIW_BORDER::_className, 0 },
    { ID_BUTTON, ZIL_VOIDF(UIW_BUTTON::New), UIW_BUTTON::_className, 0 },
The UIW_ICON is the only object not already in the object table so it is the only object
that needs to be added.
    { ID_ICON, ZIL_VOIDF(UIW_ICON::New), UIW_ICON::_className, 0 },
    { ID_MAXIMIZE_BUTTON, ZIL_VOIDF(UIW_MAXIMIZE_BUTTON::New),
      UIW_MAXIMIZE_BUTTON::_className, 0 },
    { ID_MINIMIZE_BUTTON, ZIL_VOIDF(UIW_MINIMIZE_BUTTON::New),
      UIW_MINIMIZE_BUTTON::_className, 0 },
    { ID_STRING, ZIL_VOIDF(UIW_STRING::New), UIW_STRING::_className, 0 },
    { ID_SYSTEM_BUTTON, ZIL_VOIDF(UIW_SYSTEM_BUTTON::New),
      UIW_SYSTEM_BUTTON::_className, 0 },
    { ID_TITLE, ZIL_VOIDF(UIW_TITLE::New), UIW_TITLE::_className, 0 },
    { ID_WINDOW, ZIL_VOIDF(UIW_WINDOW::New), UIW_WINDOW::_className, 0 },
    { ID_END, ZIL_NULLP(void), ZIL_NULLP(ZIL_ICHAR), 0 }
};
UI_ITEM *UI_WINDOW_OBJECT::objectTable = _objectTable;

```

Notes:

- After combining all .CPP files you will only need to reference the proper .DAT file in your application.
- If you are referencing resources out of default storage you will need to delete the default storage and create a new default storage with the correct .DAT file so that the default storage does not reference the incorrect .DAT file.
- It may also necessary to combine both of the .HPP files similar to what is being done with the .CPP files.

5.5 Security on a .DAT file

ZD-TN2014

Keywords: Security, .DAT file, Designer, MAGIC_NUMBER
Versions: 3.5+
Components: Library
Platforms: All
Issued: April 17, 1995

Question

How can I secure my .DAT file to prevent anybody with a Designer from opening the file?

Answer

To place security on a .DAT file, change MAGIC_NUMBER, located near the top of Z_STORE.HPP. If the MAGIC_NUMBER in the .DAT file does not match the MAGIC_NUMBER in the library, the .DAT file will not be opened.

When a .DAT file is opened, the library checks MAGIC_NUMBER. If you change MAGIC_NUMBER in your library, you will need to change the magic number in your .DAT files to open them. The following is a program that will change the magic number in your .DAT files by patching them.

```
#include <stdio.h>
#include <fcntl.h>
#include <io.h>
#include <ui_gen.hpp>
#include <z_store.hpp>
// This number must match exactly the number in Z_STORE.HPP.
#define NEW_MAGIC 0x47

int main(int argc, char **argv)
{
    argc--; argv++;
    while (argc)
    {
        int fd = open(*argv, 0_RDWR);
        ZINC_SIGNATURE tmp;
        read(fd, &tmp, sizeof(tmp));
        tmp.magicNumber = NEW_MAGIC;
    }
}
```

```
    lseek(fd, 0, 0);  
    write(fd, &tmp, sizeof(tmp));  
    close(fd);  
    argc--; argv++;  
}  
return 0;  
}
```

To edit an existing .DAT file in the Designer or create a new .DAT file, compile the Designer with the newly compiled library and patch the MAGIC_NUMBER of the Designer .DAT files. You can patch the Designer .DAT files in each individual subdirectory under the Designer directory, or patch these in the ZIN\BIN directory after you have built the Designer. The Designer .DAT files are all copied into the BIN directory as .ZNC files for graphics mode or .Z_T for DOS text mode.

After changing the MAGIC_NUMBER, if an application fails to come up on start-up, or all you hear is a beep, the magic number in the .DAT file probably does not match that of the library. Check the MAGIC_NUMBER in Z_STORE.HPP and the program used to change the MAGIC_NUMBER and make sure that the numbers exactly match. If one defined as a HEX value you must also define the other as a HEX value.

5.6 Loading a derived object from a .DAT file

ZD-TN3010

Keywords: .DAT, derived object, load
Versions: 3.5 and later
Components: Library
Platforms: All
Issued: October 31, 1994
File: MY_BTN.ZIP

Question

How do I load a derived button from a .DAT file?

Answer

To load a derived object from a .DAT file you will need to create a static **New()** function for that class that will call the persistent constructor for that class, which also must be provided.

Example

```
class MY_BUTTON:public UIW_BUTTON
{
public:
    MY_BUTTON(const char *name, UI_STORAGE directory, UI_STORAGE_OBJECT *file) :
        UIW_BUTTON(name, directory, file) {};

    static UI_WINDOW_OBJECT *New(const char *name,
        UI_STORAGE *directory = NULL, UI_STORAGE_OBJECT *file = NULL)
    { return(new MY_BUTTON(name, directory, file));}
```

The above code will load a derived button that has been stored in a .DAT file using OpenZinc Designer. It will work for loading objects derived from any OpenZinc class.

5.7 Using a table created in OpenZinc Designer ZD-TN3011

Keywords: table
 Versions: 4.0
 Components: Library, **.DAT** file
 Platforms: All
 Issued: November 8,1994

Question

How do I put information into a table that was created in the OpenZinc Designer?

Answer

There are two different approaches to using a table object designed in the OpenZinc Designer.

Set the record size in OpenZinc Designer and use **DataSet()** to set the data, number of records, and maximum number of records. You will also need to set the user function for the table record, because the field to set it is not currently available in the designer.

```
UIW_TABLE *table = (UIW_TABLE *) win->Get("TABLE");
table->DataSet(data, numberOfRecords);
UI_WINDOW_OBJECT *tableRecord = table->First();
tableRecord->userFunction = TableRecordCallBack;
```

In **TableRecordCallBack()**, *event.data* is a pointer to the data associated with that table record. The **UIW_TABLE** maintains an internal array of data. Typecast the pointer into a pointer of the structure type that you are using. The *event.rawCode* will contain the number of the table record that is calling the callback function.

```
EVENT_TYPE TableRecordCallBack(UI_WINDOW_OBJECT *object, UI_EVENT &event, EVENT_TYPE
ccode)
{
    switch (ccode)

    case S_SET_DATA:

        MY_DATA *myData = (MY_DATA *)event.data;
        object->Get("NAME")->Information(I_SET_TEXT, myData->name);

    }
    case S_NON_CURRENT:
    case L_SELECT:
    {
        MY_DATA *myData = (MY_DATA *)event.data;
        object->Get("NAME")->Information(I_COPY_TEXT, myData->name);
    }
}
```

Using a table created in OpenZinc Designer ZD-TN3011

```
    }  
}
```

Set the *WOF_NO_ALLOCATE_DATA* flag and maintain the data yourself.

```
UIW_TABLE *table = (UIW_TABLE *) win->Get("TABLE");  
table->woFlags |= WOF_NO_ALLOCATE_DATA;  
table->DataSet(0, numberOfRecords);
```

In **TkbleRecordCallback()** *event.data* is a pointer to the data associated with that table record. The **UIW_TABLE** maintains an internal array of data. Since the *WOF_NO_ALLOCATE_DATA* flag was set *event.data* will be 0. You will need to cast the pointer into the structure type that you are using. The *event.rawCode* will contain the number of the table record that is calling the callback function. In the following example *_myData* is a global array that has been allocated.

```
EVENT_TYPE TableRecordCallback(UI_WINDOW_OBJECT *object, UI_EVENT &event, EVENT_TYPE  
    ccode)  
{  
    switch (ccode)  
    {  
        case S_SET_DATA:  
            object->Get("NAME")->information(I_SET_TEXT,  
                _myData[event.rawCode].name);  
        }  
        case S_NON_CURRENT:  
        case L_SELECT:  
            object->Get("NAME")->information(I_SET_TEXT,  
                _myData[event.rawCode].name);  
        }  
    }  
}
```

Note:

- If the record size is set incorrectly in the .DAT file and you use DataSet() to initialize the table, then you could get unexpected results. For this reason OpenZinc recommends using method 2. To determine the size of your data structure, you could use a simple program that prints out the size of the structure. If you are doing this on multiple platforms and the size of the structure is different on each plat-

form you could use `#ifdefs` to make the structures the same across platforms. This is only important if you are setting the data's size and aren't using the "don't allocate data" flag. *OpenZinc is currently improving the ability to set the record size, and it will probably not be read in from the .DAT file.*

- Access to the `WOF_NO_ALLOCATE_DATA` flag will probably be added in a future revision.
- The data array is to be maintained by the programmer, and the programmer is responsible for freeing any memory that they have allocated. The programmer does have the opportunity to use a file of records to provide virtual memory, in which case the above code for `S_SET_DATA` would contain a read from the data file and the `S_NON_CURRENT` would store the changes back out to the file.

5.8 Checking for errors when reading .DAT files ZD-TN2007

Keywords: WOS_READ_ERROR, DAT file
Versions: 3.5 or later
Components: Library
Platforms: All
Issued: February 1, 1995

Question

How do I check for an error in reading an object from a .DAT file and exit gracefully if there is an error?

Answer

To check for an error when reading an object from a .DAT file, check the *WOSJREADJERROR* status. If this status is set, then the application can clean up and exit normally. This status is set if the .DAT file could not be found, if the resource could not be found in the .DAT file, or if any object in the resource had a problem being read in. If the *WOS_READ_ERROR* status is set, you could call error system to display a message, explaining that there was an error reading the resource from the .DAT file. Then the program can exit. The following is an example:

```
UI_WINDOW_OBJECT::errorSystem = new UI_ERROR_SYSTEM;
UIW_WINDOW *window = new UIW_WINDOW("test.dat~WINDOW_1");
if (FlagSet(window->woStatus, WOS_READ_ERROR)
{
    window->errorSystem->ErrorMessage(windowManager, WOS_NO_STATUS, "There was a
        problem loading the window from the .DAT file", ".DAT FILE ERROR");
    delete window;
    delete UI_WINDOW_OBJECT::errorSystem;
    return(0);
}
```

If the window is read in, but a *WOS_READ_ERROR* is generated, you can traverse the list of the window and check each object's *WOS_READ_ERROR* status to determine which object caused the problem.

5.9 Deriving an object in OpenZinc Designer ZD-TN2009

Keywords: Designer, deriving object

Versions: 4.0

Components: Designer

Platforms: All

Issued: February 3, 1995

Question

How do I derive an object in OpenZinc Designer?

Answer

To derive an object in OpenZinc Designer, first place an object on the window. Open the edit window for that object and select the advanced page. In the field for Derived Name, replace <none> with the name of the derived class. For example, if a class called MY_BUTTON was derived from UIW_BUTTON, replace <none> with MY_BUTTON.

When saving a file, the Designer generates a .CPP and a .HPP file, and you must include the header file with the derived class declaration in the generated .CPP file. For example:

```
#include <ui_win.hpp>
#define USE_DERIVED_OBJECTS
#include "test.hpp"
void z_jump_dumrny-(void) { } // Bug fix for broken linkers.
//Line added to include header file which contains the derived class declaration.
#include "my_btn.hpp"

static UI_ITEM_userTable[] =
```

Remember that because the Designer generates a new .CPP file each time a .DAT file is saved, you must edit the .CPP file. You can prevent this by selecting Preferences under File on the Designer's main window and deselecting Write CPP.

Next, type in the class declaration, which must contain entries for the required persistent functions. These functions are a persistence constructor and a static New() function. The following is an example of a class declaration:

```
class MY_BUTTON:public UIW_BUTTON
{
public:
    MY_BUTTON(const ZIL_ICHAR *name, ZIL_STORAGE_READ_ONLY *directory, ZIL_STORAGE_OBJECT_
READ_ONLY *file) :
        UIW_BUTTON(name, directory, file) {};
    static UI_WINDOW_OBJECT *New(const ZIL_ICHAR *name,
```

Deriving an object in OpenZinc Designer ZD-TN2009

```
ZIL_STORAGE_READ_ONLY *directory = ZIL_NULLP(ZIL_STORAGE_READ_ONLY), ZIL_STORAGE.  
OBJECT_READ_ONLY *file = ZIL_NULLP(ZIL_STORAGE_OBJECT_READ_ONLY))  
    { return(new MY_BUTTON(name, directory, file));}  
    EVENT_TYPE Event(const UI_EVENT &);  
};
```

5.10 Designer fails to launch ZD-TN2011

Keywords: Designer, .ZNC, .Z_T
Versions: 4.0
Components: Designer
Platforms: All
Issued: February 8, 1995

Question

Why does the Designer fail to launch, and why does it only beep?

Answer

The Designer will not come up if it can not find its .ZNC or .Z_T files. In some environments it will bring up an error message saying that it could not find a .ZNC or .Z_T file, however in other environments it will only beep and close the Designer down. After the Designer has been built, these files will be found in the OpenZinc/BIN directory. If these files are in the same directory as the Designer executable it should have no problems finding them. If they are in different directories you can create an environment variable, OpenZinc_PATH, which contains the path to the .ZNC or .Z_T files.

The .ZNC files contain the resources used by the Designer when run in graphics mode. The .Z_T files contain the resources used when running the Designer in text mode. It is not necessary to have separate .DAT files for text and graphics mode. The Designer has separate .DAT files so that when running in text mode the resources are positioned proportionally correct.

5.11 Designer import and export file types

ZD-TN2012

Keywords: Designer, import, export, .DAT, .TXT, .BMP, .ICO, .XPM
Versions: OpenZinc 4.0
Components: Designer
Platforms: All
Issued: February 14, 1995

Question

What types of files can I import into and export out of each service in the Designer?

Answer

You can import and export the following file types:

Window Editor

- .DAT
- .TXT

Image Editor

- .BMP
- .ICO
- .XPM
- .DAT

Note: You can only import and export the file type for the specific platform you are running the Designer in. For example you can not import a Windows .BMP into the Designer when running in OS/2. It would need to be a OS/2 .BMP file.

Help Editor

- .DAT
- .TXT

Message Editor

.DAT

5.12 Displaying information in a table created in the Designer ZD-TN3011

Keywords: table, Designer
 Versions: 4.0 Release 1
 Components: Library, .DAT file
 Platforms: All
 Issued: November 8, 1994

Question

How do I display information in a table that was created in the OpenZinc Designer?

Answer

There are two different approaches to displaying information in a table object designed in the OpenZinc Designer. The first method is to use the WOF_NO_ALLOCATE_DATA flag and the second method is to allow the table to allocate the associated data. The WOF_NO_ALLOCATE_DATA flag is used to tell the table object not to allocate internal memory to track the information presented in its records. For the callbacks of both methods event.data will be a void pointer to the memory allocated (if any was allocated) by the table for the information, and eventrowCode will be the index of the record being accessed. Access to the WOF_NO_ALLOCATE_DATA flag is accessible in OpenZinc Designer with revision 2.

Method 1: Using the WOF_NO_ALLOCATE_DATA flag:

The following code initializes the table to have a maximum of 15 records and to start with 15 records.

```
UIW_TABLE *table1 = (UIW_TABLE *)win->Get("NO_ALLOCATE_TABLE");
if (table1)
{
    table1->woFlags |= WOF_NO_ALLOCATE_DATA; //This should be set in the Designer
        // after revision 2.
    table1->DataSet(ZIL_NULLP(void), 15, 15);
    _tableData = new ZIL_ICHAR *[15];
    for (int i = 0; i<15; i++)
    {
        _tableData[i] = new ZIL_ICHAR[15]; // _tableData is a global variable
        sprintf(_tableData[i], "String%d", i);
    }
}
```

In the callback associated with this table, NoAllocateCallback, the WOF_NO_ALLOCATE_DATA flag is set so eventdata will be NULL, and eventrowCode will contain the index of the record being handled by the callback function.

```
EVENT_TYPE NoAllocateCallback(UI_WINDOW_OBJECT *object, UI_EVENT &event, EVENT_TYPE
    ccode)
{
```

Displaying information in a table created in the Designer ZD-TN3011

```
switch (ccode)
{
case S_SET_DATA:
    UI_WINDOW_OBJECT *winObject = object->Get("STRING");
    winObject->Information(I_SET_TEXT, tableData[event.rawCode]);
}
break;

case L_SELECT:
case S_NON_CURRENT:
    UI_WINDOW_OBJECT *winObject = object->Get("STRING");
    winObject->Information(I_COPY_TEXT, _tableData[event.rawCode]);
}
break;
}

return 0;
}
```

However, with the method of using the WOF_NO_ALLOCATE_DATA flag any memory allocated to keep track of information related to the table must be maintained by the programmer. In a database application, one method of handling data would be to use a file of records to provide virtual memory, in which case the above code for S_SET_DATA would contain a read from the data file and the S_NON_CURRENT would store the changes back out to the file.

Method 2: Letting the table allocate the data.

Set the record size in the OpenZinc Designer and use the DataSet function to set the data, number of records and maximum number of records. The following code initializes the table to have a maximum of 15 records and to start with 15 records. It also initializes the data associated with the table.

```
UIW_TABLE *table2 = (UIW_TABLE *)win->Get("ALLOCATE_TABLE");
if (table2)
{
    ZIL_ICHAR table2Data[15][15];
    for (int i = 0; i<15; i++)
    {
        sprintf(table2Data[i], "String_%d", i);
    }
    table2->DataSet(table2Data, 15, 15);
}
```

In **AllocateCallback()** event.data will be a pointer to the data that is associated with that table record. The UIW_TABLE maintains an internal array of data. Typecast the pointer into a pointer of the structure type that you are using.

Displaying information in a table created in the Designer ZD-TN3011

```
EVENT_TYPE AllocateCallback(UI_WINDOW_OBJECT *object, UI_EVENT &event, EVENT_TYPE
    ccode)
{
    ZIL_ICHAR *string = (ZIL_ICHAR *)event.data;

    switch (ccode)
    {
    case S_SET_DATA:
        {
            UI_WINDOW_OBJECT *winObject = object->Get("STRING");
            winObject->Information(I_SETJEXT, string);
        }
        break;

    case L_SELECT:
    case S_NON_CURRENT:
        {
            UI_WINDOW_OBJECT *winObject = object->Get("STRING");
            winObject->Information(I_COPYJEXT, string);
        }
        break;
    }

    return 0;
}
```


Using the ZAF_STRING_EDITOR class of the Designer ZD-TN4025

5.13 Using the ZAF_STRING_EDITOR class of the Designer ZD-TN4025

Keywords: String editor
Version: 4.0
Component: Designer; ZAF_STRING_EDITOR
Platforms: ALL
Issued: January 31, 1995

Question

How can I use the String Editor outside of the Designer?

Answer

The String Editor of the Designer is an instance of the **ZAF_STRING_EDITOR** class. The file that contains the source code for the **ZAF_STRING_EDITOR** is found in **OpenZinc\design\stredit\stredit.cpp**. Also in this directory are the makefiles and a sample program module, **main.cpp**, to build an executable program that uses the String Editor. Since the String Editor depends on functions found in the **ZAF_SERVICE_MANAGER** class, build **service.lib** by running the appropriate makefile from **OpenZinc\design**.

A look at **main.cpp** from **OpenZinc\design\stredit** shows the following:

```
#include "stredit.hpp"
int UI_APPLICATION: -.Main(void)
{
    UI_APPLICATION::LinkMain();
    _serviceManager = new ZAF_SERVICE_MANAGER;

    // Create and add the support editors.
    ZAF_STRING_EDITOR *stringEditor = new ZAF_STRING_EDITOR;
    stringEditor->Information(I_ACTIVATE_SERVICE, ZIL_NULLP(void) );

    // Process user responses.
    UI_APPLICATION::Control();

    // Clean up.
    return (0);
}
```

Familiarize yourself with these main components of the String Editor and their use:

ZAF_SERVICE_MANAGER

The *_serviceManager* is declared in **SERVICE.HPP** in **OpenZinc/design/service.SERVICE.HPP** is included in **STREDIT.HPP**. Create *_serviceManager* first, the **ZAF_STRING_EDITOR** class references it. The *_serviceManager* is responsible for managing the persistent object storage file for the String Editor, **p_stredit znc**.

ZAF_STRING_EDITOR

The constructor of **ZAF_STRING_EDITOR** creates the String Editor window in memory. It also creates and adds a string driver device to the *eventManager*. The purpose of the string driver device is to monitor the event queue, and when an F12 key is pressed, the String Editor service is restored from its iconized state. This occurs with the following line of code found in the **Poll()** function of the string driver device:

```
Information(I_ACTIVATE_SERVICE, ZIL_NULLP(void));
```

After the computer executes the above line of code, the current editable object on the screen, if any, is found. Then, with the following line of code found in the **Poll()** function of the string driver device, the current editable object is set to the object that gets String Editor service:

```
Information(I_SET_REQUESTOR, requestor);
```

I_ACTrVATE_SERVICE—Information request

The purpose of the line

```
stringEditor->Information(I_ACTIVATE_SERVICE, ZIL_NULLP(void));
```

is to add the string edit window created in the **ZAF_STRING_EDITOR** constructor to the *windowManager*. If it already exists in the *windowManager*'s list, this will make the string edit window become the current window. Also, if it was previously minimized, it gets restored to its normal size.

I_SET_REQUESTOR—Information request

When the user presses the F12 key, the string driver detects this and gets a pointer to the current window if one exists. It then searches the current window for an editable object that is current. If it finds one, it assigns this pointer as the requestor of the string editor service with the following line of code:

```
stringEditor->Information(I_SET_REQUESTOR, requestor);
```

After this occurs, the user can then select the characters in the table that are to appear in the requestor's text field.

If the user then selects the OK button on the string edit window, the text that the user has entered in the "Current" field of the string editor gets set as the actual text for the requestor object. After the text for that object gets set, the string edit window gets minimized if its *WOAF_LOCKED* flag is set. If this flag is not set, the string edit window is closed.

Using the ZAF_STRING_EDITOR class of the Designer ZD-TN4025

The only minor modification you may want to make to **stredit.cpp** is to increase the max length of the **UIW_EDIT_STRING** class from 100 to 1024. This is helpful when editing large text fields. The string editor requires the **p_string.znc** persistent object file for retrieval of the string editor window.

In addition to your **.DAT** file, if any, **p_string.znc** must accompany your application. If you do not want two persistent object storage files with your application, you can take a copy of this file and use the **Window I Import** selection to add your resources to this file. When doing this you would need to link in **p_string.cpp** to your application.

The string edit window defaults to code page ISO 8859-1 in Unicode mode. The code page control is nonselectable unless you are using the Unicode Designer that comes with the Unicode key. With code page control enabled, you can access fonts from the Unicode character set, including Japanese, Chinese, and the Korean font set.

An example using the **ZAF_STRING_EDITOR** outside of the Designer can be found on the bulletin board, user main conference, Tech Support listing.

Section 6

UI objects and programming techniques

6.1 Removing geometry constraints ZD-TN1014

Keywords: Geometry management, constraints
Versions: 4.0+
Components: UI_GEOMETRY_MANAGER, UI_CONSTRAINT
Platforms: All
Issued: May 15, 1995

Question

If I delete an object that has geometry constraints, my program doesn't continue to function properly. What can I do to fix this?

Answer

In this technote we will discuss how to remove any geometry constraints associated with an object.

Because OpenZinc has defined geometry management so that there is no extra overhead in the UI_WINDOW_OBJECT class, you must delete any geometry constraints associated with the object.

First we must get a pointer to the geometry manager.

```
UI_GEOMETRY_MANAGER *geometryManager = (UI_GEOMETRY_MANAGER *)Get(NUMID_GEOMETRY);
```

This example maintains the pointer as a member of the controlling class. When you delete an object that is managed, search through all the constraints and compare the constraint's object with the object that you are deleting. If they are equal remove that particular constraint. You can have multiple geometry constraints for any given object. All of these constraints must be removed or your program will terminate abnormally.

Here is a code snippet that will search for and delete any geometry constraints associated with an object.

```
// Get a pointer to the object to be deleted.
UI_WINDOW_OBJECT *button = Get("RIGHT_BOTTOM");

// Maintain a separate pointer for comparison.
UI_WINDOW_OBJECT *constObj;

// Get the first constraint.
UI_CONSTRAINT *geoConst = geometryManager->First();

// Continue while there are constraints.
while (geoConst)
{
    // Get the next constraint.
    UI_CONSTRAINT *nextConst = geoConst->Next();
    geoConst->Information(I_GET_OBJECT, &constObj);
}
```

Removing geometry constraints ZD-TN1014

```
// Compare the objects.
if (constObj == button)
{
    // Delete this constraint.
    *geometry-Manager - geoConst;
    delete geoConst;
}
geoConst = nextConst;

// Now we can safely delete the object.
*this - button;
delete button;

// Redisplay the window to reflect the deletion of the object.
Event(S_REDISPLAY);
```

When you delete an object that has been managed, you must delete any constraints that are associated with that object.

6.2 Using a button to close a window

ZD-TN2000

Keywords: close window, button
Version: 3.5, 3.6,4.0
Component: Library
Platforms: All
Issued: July 14, 1994

Question

How do I close a window using a button attached to that window?

Answer

You may close a window using a button three different ways.

If the button is only used to close the window and performs no other action, you need only set the *BTF_SEND_MESSAGE* flag on the button and set the button's value equal to *S_CLOSE*, or in the designer set the value equal to -11.

If the button is to perform some other function besides closing the window, and the window is a derived window, it can still be a *BTF_SEND_MESSAGE* button. Just set the value of the button equal to a user-defined event. When pressed the button will place the user defined event on the queue and it will be passed down to the window. The window can then trap this event in its **Event()** function, perform any necessary action and then place an *S_CLOSE* on the queue with the following code.

```
eventManager->Put(S_CLOSE);
```

If a user function has been attached to the button, before leaving the user function place an *S_CLOSE* on the queue with the following line of code:

```
object->eventManager->Put(S_CLOSE);
```

Do not subtract the window from the *windowManager* while in the user function. If you do, your application will crash. This is because you are removing the object that called the user function and you must be able to return to that object.

If the window is an MDI child you should use *S_MDICHILD_EVENT + S_CLOSE* (-511 in the Designer). The *S_MDICHILD_EVENT* message instructs the *windowManager* that this message needs to be handled by the MDI parent.

The first two methods are preferred when using a button to perform an action because they take advantage of OpenZinc's event driven architecture.

6.3 Changing object flag or status at run time

ZD-TN2002

Keywords: flag, status, run time
 Version: 3.5,3.6
 Component: Library
 Platforms: All
 Issued: July 7, 1994

Question

How do I change the flag or status of an object at run time?

Answer

There are two methods that can be used to change a status or a flag on an object. One is to use the **Information()** function. The other is to mask the specific bit on or off. Following is an example of setting and unsetting the selected status of a button.

Method 1 (Information() function):

```
UIS_STATUS status = WOS_SELECTED;
button->Information(SET_STATUS, &status, ID_WINDOW_OBJECT);
button->Information(CHANGED_STATUS, NULL);
button->Information(CLEAR_STATUS, &status, ID_WINDOW_OBJECT);
button->Information(CHANGED_STATUS, NULL);
```

When using the **Information()** function the proper *objectID* must be passed in or it will not work. Consult the *Programmer's Reference* for the object's **Information!** function to determine the proper *objectID* to be used with the specific flag or status you wish to change. When changing the flag of an object *UIF_FLAGS* should be used in place of *UIF_STATUS*.

Method 2 (Bit masking):

Mask On

```
button->woStatus |= WOS_SELECTED;
button->Information(CHANGED_STATUS, NULL);
```

Mask Off

```
button->woStatus &=~WOS_SELECTED;
button->Information(CHANGED_STATUS, NULL);
```

Whenever a status or a flag of an object has been changed the object must be told that it has been changed. This is done with the **Information!** function and passing in a *CHANGED_STATUS* if a status was changed or passing in a *CHANGED_FLAGS* if a flag was changed.

Check boxes or toggle buttons on a window ZD-TN2004

6.4 Check boxes or toggle buttons on a window ZD-TN2004

Keywords: check box, toggle, button

Versions: 3.5 and later

Components: Library

Platforms: All

Issued: September 13, 1994

Question

When I select a check box or toggle button on my window I cannot deselect it.

Answer

When using check boxes or toggle buttons the *WNF_SELECT_MULTIPLE* flag must be set on the parent. If this flag has not been set it will cause the check boxes or toggle buttons to behave as radio buttons.

This flag can be set on the following parent objects:

UIW_WINDOW

UIW_GROUP

UIW_VT_LIST

UIW_HZ_LIST

UIW_POP_UP_MENU

6.5 Nonselectable gray and nonselectable black ZD-TN2013

Keywords: nonselectable gray, nonselectable black
 Versions: 3.5+
 Components: Library
 Platforms: All
 Issued: April 17, 1995

Question

How do I make an object nonselectable gray or nonselectable black programmatically?

Answer

To make an object nonselectable gray, set the WOF_NON_SELECTABLE flag on the object.

```
object->woFlags |= WOF_NON_SELECTABLE;
object->Information(I_CHANGED_FLAGS, NULL); //For versions earlier than 4.0 replace
//I_CHANGED_FLAGS with CHANGED_FLAGS.
```

To make an object nonselectable black, set the WOAF_NON_CURRENT and WOF_VIEW_ONLY flags.

```
object->woFlags |= WOF_VIEW_ONLY;
object->woAdvancedFlags |= WOAF_NON_CURRENT;
object->Information(I_CHANGED_FLAGS, NULL); //For versions earlier than 4.0 replace
//I_CHANGED_FLAGS with CHANGED_FLAGS.
```

The aforementioned method only works on editable objects such as a UIW_STRING object. If you want to make a noneditable object, such as button, nonselectable black:

- Derive your own button object.
- Set the WOAF_NON_CURRENT flag on the object.
- In the derived Event() function trap for the L_SELECT and L_BEGIN_SELECT messages. This will prevent them from being passed to the base class.

6.6 Scrolling more than 32,767 records in a UIW_TABLE ZD-TN2016

Keywords: UIW_TABLE, scrolling
Versions: 4.0+
Components: Library
Platforms: All
Issued: May 10, 1995

Question

How can I scroll more than 32,767 records in a table?

Answer

You can scroll more than 32,767 records by controlling the scrolling of the table. To control the scrolling of the table, derive your own table class from UIW_TABLE. The class will require both an Event() function and a ScrollEvent() function.

The Event() function will need to process the S_VSCROLL event. The S_VSCROLL event is sent to the table when the scroll bars position is changed. It should be passed to the ScrollEvent() function. You may also want to process the L_UP, L_DOWN, L_PGUP, L_PGDN, L_TOP, and L_BOTTOM to handle scrolling from the keyboard. The L_ events are mapped from keyboard events, in the Event() function of an object. When the Event() receives them it should translate them to a S_VSCROLL with the scroll.delta set to the proper value. Then it should send them to the ScrollEvent() function.

The table should only contain the number of records that are visible. This makes it easier to control the scrolling of the table. If the table only contains the number of records that are visible, call DataSet() to change the data in the records. Otherwise you must worry about scrolling the records and updating the data.

The information that the scroll bar maintains for scrolling is a ZIL_INT16, a 16-bit int. Because of this, the scroll bar can only scroll through 32,767 positions. To overcome this, set up a ratio between the number of objects you want to scroll and the number that can actually be scrolled. For example if you wanted to scroll 320,000 records in your table you could set the scroll bar's maximum to be 32,000 and scroll 10 records with each position the scroll bar moves.

After the table's Event() function has received an S_CREATE, set the scroll information on the table's scroll bar. To do this create a UI_SCROLL_INFORMATION structure and send it to the scroll bar's Event() function. You may want to initialize the scroll information like this:

```
UI_SCROLL_INFORMATION scrollInfo;  
scrollInfo.current = 0;  
scrollInfo.minimum = 0;  
scrollInfo.maximum = 32000; // 320000 divided by 10  
scrollInfo.showing = 10;  
scrollInfo.delta = 1;
```

Scrolling more than 32,767 records in a UIW_TABLE ZD-TN2016

When ScrollEvent() receives a S_VSCROLL, the scroll bar's current position and the data in the table must be updated. The scroll bar's new position can be calculated by getting the scroll bar's current position. Do this by calling the scroll bar's Information() function, with I_GET_VALUE. The event.scroll.delta, passed into the ScrollEventO function, contains the distance the scroll bar's thumb button was moved. Adding this to the current value will give the scroll bar's new position, which will be used to calculate what data should be displayed in the table. Then set the table data with a DataSet() and set the scroll bar's scroll information by creating a new UI_SCROLL_INFORMATION structure that contains the scroll bar's new position as the current position. Then send it to the scroll bar's EventO function, along with S_REDISPLAY. Update the data in the table before the new scroll information is sent to the scroll bar, otherwise the DataSet() on the table will reset the scroll information on the scroll bar.

Columns in lists ZD-TN3000

6.7 Columns in lists

ZD-TN3000

Keywords: column, list
Versions: 3.5 and later
Components: Library
Platforms: All
Issued: July 20, 1994
File: **AGENCY.ZIP**

Question

How do I get columns in a vertical list?

Answer

Derive a new class from **UIW_BUTTON**, **UIW_STRING**, or **UI_WINDOW_OBJECT** and provide your own **DrawItem()** function.

In the **DrawItem()** function, use the appropriate display member functions to display the data relative to the `trueRegion` coordinates of the derived object. In this way you can ensure that your objects display in a fixed width that will allow columns.

Note

OpenZinc Application Framework 4.0 supports a table object that aligns objects in columns automatically. This may be more appropriate for some applications.

6.8 Retrieving the nth item from a list

ZD-TN3001

Keywords: combo box, list, nth item
 Version: 3.0 and later
 Component: Library
 Platforms: All
 Issued: July 8, 1994

Question

How do I get a pointer to the element in my list—a window, vertical list, horizontal list—or in my combo box?

Answer

There are two different techniques for getting a certain element from a list depending on which type of list you are using.

To retrieve the third item from a **UIW_WINDOW**, **UIW_VT_LIST**, or **UIW_HZ_LIST**, execute the following code with a pointer called *list*:

```
UI_WINDOW_OBJECT *item;
item = list->UI_LIST::Get(2); // The lists are zero based.
```

This method will not work with a combo box. For a combo box you must use method 2.

To retrieve the third item from a **UIW_COMBO_BOX**, use the **Get()** function. For example to get the third element in a combo box with a pointer called *comboBox* execute the following code:

```
UI_WINDOW_OBJECT *item;
item = comboBox->Get(2); // The lists are zero based
```

6.9 Adding an object at run time

ZD-TN3002

Keywords: list, object, adding, run time
 Version: 3.0 and later
 Component: Library
 Platforms: All
 Issued: July 6, 1994

Question

How do I add an object to a list at run time? When I do, it doesn't show up on the screen.

Answer

To add an object to a list and have it show up, simply add the item(s) and call the list's **Event()** function with an *S_REDISPLAY* message.

The object doesn't show up on the screen after you add it to a list because of a Zinc design principle. After adding an item to a list, the list will not redisplay itself until it receives an *S_REDISPLAY*. This provides for greater efficiency because it eliminates unnecessary (and slow) screen updates when adding several items to a list.

```
EVENT_TYPE AddToList(UI_WINDOW_OBJECT *object, UI_EVENT    EVENT_TYPE ccode)
{
    if (ccode != L_SELECT)
        return ccode;
    UI_WINDOW_OBJECT *list = object->parent->Get("VLIST");

    if (list)
    {
        // Create the string to be added.
        UIW_STRING *string = new UIW_STRING(0,0,0,"TEST");
        // Create the event that will add the string to the vertical list.
        UI_EVENT tEvent(S_ADD_OBJECT);
        tEvent.data = string;
        list->Event(tEvent);

        // Redisplay the list
        tEvent.type = S_REDISPLAY;
        list->Event(tEvent);
    }
    return ccode;
}
```

This technique works equally well with other objects such as windows or combo boxes. The call to **Event()** in each of these cases is exactly the same.

6.10 Moving an object at run time

ZD-TN3003

Keywords: object, run time
 Version: 3.5 and later
 Component: Library
 Platforms: All
 Issued: July 12, 1994
 File: None

Question

How do I move an object on a window at run time?

Answer

Simply modify the relative coordinates of the object and then call the **Information()** function for that object with a message of *CHANGED_FLAGS*. This will cause the object to update its information.

Example

```
EVENT_TYPE MY_WINDOW::Event(const UI_EVENT &event)

// Get a pointer to the button
UI_WINDOW_OBJECT *button = Get("THIS_BUTTON");

int new_height = button->trueRegion.bottom - button->trueRegion.top;
int string_width = display->TextWidth("Longest Button Text") /
  display->cellWidth;

// Calculate its new coordinates
// 3/4 of the way across the screen
button->relative.left = (trueRegion.right - trueRegion.left) * 3/4 *
  display->cellWidth;
// 3 cells from the bottom
button->relative.top = (trueRegion.top - trueRegion.bottom - 3) *
  display->cellHeight;
button->relative.right = button->relative.left +
  ((string_width + 2) * display->cellWidth);
button->relative.bottom = button->relative.top + new_height;

// Cause the button to reposition and redisplay itself.
button->Information(CHANGED_FLAGS, NULL);

return (ccode)
```


Removing the title from a window with the Designer ZD-TN3004

6.11 Removing the title from a window with the Designer ZD-TN3004

Keywords: window, title
Version: 3.0 and later
Component: Library
Platforms: DOS, Windows, OS/2, Motif
Issued: July 15, 1994

Question

How do I remove the tide from a window created in the Designer?

Answer

Using the pointer to the window, get a pointer to the title with the **Get()** function. Then subtract the title from the window and delete it.

```
UI_WINDOW_OBJECT *title = window->Get(NUMID_TITLE);

*window
- title;

delete title;
```

Note:

While this technique should compile in all environments, some environments, such as Macintosh, do not allow windows without titles. In these cases the code would have no effect. Since this is not supported in all environments you should avoid the practice in order to create applications that are feature-portable as well as code-portable.

6.12 Selecting a check box or radio button programmatically ZD-TN3005

Keywords: check box, radio button
Version: 3.0 and later
Component: Library
Platforms: All
Issued: July 13, 1994

Question

How do I cause a check box or radio button to be selected?

Answer

To cause the check box to be selected set the *WOS_SELECTED* flag in the *woStatus* member of the check box. Then call its information function with *CHANGED_STATUS*. This will cause it to redisplay itself with the new status.

```
checkBox->woStatus |= WOS_SELECTED;  
checkBox->Information(CHANGED_STATUS, NULL);
```

To cause a radio button to be selected just read it to its group. You will need to have the *WNF_AUTO_SELECT* flag set on the group that contains the radio buttons.

```
*group  
+ radioButton;
```

6.13 Lifetime of an object

ZD-TN3006

Keywords: messages, object
Versions: All
Components: Library
Platforms: All
Issued: August 4, 1994

Question

What messages does an object receive during its lifetime?

Answer

An object goes through the following steps during its lifetime:

When the object is first created, the object's constructor is called. Before it does anything it calls its base class constructor(s). If the constructor called is the persistent constructor, it performs the following steps:

- It calls its nonpersistent base class constructor(s).
- It loads in its information from the data file.
- It directly calls the **Information()** function(s) of its base class(es) with an *INITIALIZE_CLASS* message. *Note:* In version 4.0 the *INITIALIZE_CLASS* message has been changed to *I_INITIALIZE_CLASS*.

The constructor then calls its own class's **Information!**) function with an *INITIALIZE_CLASS* message. This initializes any information that is not dependent on the object having a *screenID* (its internal flags, its *numberID*, etc.) *Note:* In version 4.0 the *INITIALIZE_CLASS* message has been changed to *I_INITIALIZE_CLASS*.

When the object is added to the Window Manager or a list that already has a *screenID*, such as a window, a vertical or horizontal list, or a group, the list sends two messages to the object:

- First, an *S_INITIALIZE* message is sent to the object. This causes the object to set up its position information. This is when the *relative* coordinates are converted (i.e. cell to graphics or vice versa). These coordinates are determined by the status of the *woStatus* flags. (If the *WOS_GRAPHICS* flag is set there is no conversion to graphics.)
- Second, the list sends the object an *S_CREATE*. This is when the screen position (i.e. *trueRegion* coordinates) is determined. This position is determined by the *relative* position of the field and the amount of available space within the list.

When the object is removed from a list (e.g., as a result of an *S_CLOSE* message in the case of a window attached to the Window Manager), we subtract the object from the list, and then send the object an *S_DEINITIALIZE* message. This informs the object that it has been removed from the screen, and it should save its

data if necessary. The object then resets its *screenID* to 0. In the case of a list object, the object passes the *S_DEINITIALIZE* event to each of its children. In the case of an *S_CLOSE* the window manager then calls the object's destructor (unless the *WOAF_NO_DESTROY* flag has been set on the window).

When the object is destroyed, the object's destructor is called; this in turn calls its base class destructor(s). At this point the destructor takes care of freeing up any memory that the object is responsible for. This includes calling the destructors for any child objects.

Note:

- An object is also sent *S_INITIALIZE* and *S_CREATE* messages if it is a child of a window that is added to the Window Manager.
- Objects may receive many other messages (or events) during their lifetimes, but these messages are common to all OpenZinc objects.

Forcing an invalid field to remain current ZD-TN3007

6.14 Forcing an invalid field to remain current ZD-TN3007

Keywords: current, invalid field, validation
Versions: 3.5, 3.6
Components: Library
Platforms: All
Issued: July 26, 1994
File: CURVALID.ZIP

Question

I want to keep an object current until it passes its validation. How do I keep a user from tabbing or mouse clicking out of it?

Answer

OpenZinc does not have this feature built in; however, you can simulate it by forcing the object to be current. To do this:

Send a user-defined event to the window to tell it to make the correct object current again. To do this create a new event that will tell the object's parent to add the object pointed to by *userObject* to itself.

```
const EVENT_TYPE READD_CHILD = 10001;
```

For the case of *S_NON_CURRENT*, in the user function for the object, set the parent's *userObject* pointer equal to the object. Put an event on the Event Manager's queue that will tell the window to re-add the object pointed to by the *userObject* pointer.

```
EVENT_TYPE myUserFunction(UI_WINDOW_OBJECT *object, UI_EVENT &, EVENT_TYPE ccode);  
  
if(!object->parent->userObject)  
    object->parent->userObject = object;  
object->eventManager->Put(UI_EVENT(READD_CHILD));
```

In order to interpret the new event you must derive a new window from *UIW_WINDOW* like this:

```
class MY_WINDOW: public UIW_WINDOW  
{  
  
};
```

When the window receives the event, it must add to itself the object pointed to by *userObject*. Then the window should set the *userObject* pointer = 0.

```
MY_WINDOW::Event(const UI_EVENT &event)
{
    case READD_CHILD:
        *this + (UI_WINDOW_OBJECT *)userObject;
        userObject = 0;
}
```

This method works equally well for vertical and horizontal lists.

6.15 Menus on the Macintosh

ZD-TN3015

Keywords: Menu, Macintosh
 Versions: 4.0 and later
 Components: Library
 Platforms: Macintosh
 Issued: March 22, 1995

Question

How do I get the menus at the top of the screen to change when I change the current window?

Answer

To swap the menus when you change the current window do the following:

First, derive a class from UIW_WINDOW with an Event() function to handle the S_CURRENT and S_NON_CURRENT messages.

When handling these messages in the Event() function subtract the menu for that window on an S_NON_CURRENT and add it to the window on an S_CURRENT. When deriving this class, provide the class definition:

```
class MY_WINDOW: public UIW_WINDOW
{
public:
    MY_WINDOW(const char *name, ZIL_STORAGE_READ_ONLY *file = NULL,
              ZIL_STORAGE_OBJECT_READ_ONLY *object = NULL);
    ~MY_WINDOW();
    virtual EVENT_TYPE Event (const UI_EVENT Seventh-
protected:
    UI_WINDOW_OBJECT *menu;
};
```

Second, provide a constructor for your class.

In this example we load the window from the .DAT file and provide a persistent constructor. In the constructor get a pointer to the menu so that we can add and subtract the menu later.

```
MY_WINDOW::MY_WINDOW(const char *name, ZIL_STORAGE_READ_ONLY *file,
                    ZIL_STORAGE_OBJECT_READ_ONLY *object): UIW_WINDOW(name, file, object)
{
    menu = Get("MENU");
}
```

Third, provide a destructor for the window.

Since we add and subtract the menu from the window, our destructor must free the memory used by the menu.

```
MY_WINDOW::~MY_WINDOW()
{
    #if defined(ZIL_MACINTOSH)
        delete menu;
    #endif
}
```

Fourth, the window's Event() function must handle the S_CURRENT and the S_NON_CURRENT events.

As we mentioned earlier, in Event() subtract the menu for S_NON_CURRENT and add it for S_CURRENT.

```
EVENT_TYPE MY_WINDOW::Event(const UI_EVENT &event)
{
    EVENT_TYPE ccode = event.type;

    switch (ccode)
    {
    #if defined(ZIL_MACINTOSH)
        case S_CURRENT:
            *this
                + menu;
            ccode = UIW_WINDOW::Event(event);
            break;
        case S_NON_CURRENT:
            *this
                - menu;
            ccode = UIW_WINDOW::Event(event);
            break;
    #endif
        default:
            ccode = UIW_WINDOW::Event(event);
    }
    return ccode;
}
```

Finally, call the constructor for your derived class instead of the UIW_WINDOW constructor.

```
UI_APPLICATION::Main()
{
    UIW_WINDOW *win1 = new MY_WINDOW("p_test.dat~RESOURCE_1");
    UIW_WINDOW *win2 = new MY_WINDOW("p_test.dat~RESOURCE_2");

    *windowManager
        + win1
        + win2;
```


Menus on the Macintosh ZD-TN3015

```
Control();  
return (0);  
}
```

6.16 Adding pop-up menus to the screen ZD-TN3016

Keywords: pop-up menus, Window Manager
 Versions: 4.0 and later
 Components: Library
 Platforms: All
 Issued: April 16, 1995

Question

I want to use a pop-up menu without attaching it to a menu bar. How do I add a pop-up menu to the screen and have it display itself without a menu bar?

Answer

To use pop-up menu without attaching it to a menu bar, create a UIW_POP_UP_MENU object and add it directly to the Window Manager.

Creating and adding a UIW_POP_UP_MENU to the Window Manager causes the pop-up to display itself without a menu bar.

To create the menu, you can use two different constructors. The first constructor takes an array of UI_ITEMS:

```
EVENT_TYPE AddPopupMenu(UI_WINDOW_OBJECT *object, UI_EVENT &, EVENT_TYPE ccode)

    if (ccode != L_SELECT)
        return (0);
    // Array of UI_ITEMS used to initialize the pop-up menu
    UI_ITEM menuItems[]=
    {
        {S_CLOSE_TEMPORARY, ZIL_NULLP(void), "Cancel",MNIF_SEND_MESSAGE},
        {0, ZIL_NULLP(void), "", MNIF_SEPARATOR},
        {L_EXIT_FUNCTION, ZIL_NULLP(void), "Exit",MNIF_SEND_MESSAGE},
        {0, CloseWindow, "Close", MNIF_SEND_MESSAGE},
        {0, ZIL_NULLP(void), 0, 0}
    };
    UIW_POP_UP_MENU *menu = new UIW_POP_UP_MENU(object->trueRegion.right, object-
>trueRegion.bottom,WNF_NO_FLAGS,menuItems);
    menu->woStatus 1= WOS_GRAPHICS;
    menu->woAdvancedFlags I= WOAF_TEMPORARY;
    *object->windowManager
    + menu;
    return (0);
```

The second constructor takes an initial position:

Adding pop-up menus to the screen ZD-TN3016

```
EVENT_TYPE AddPopupMenu2(UI_WINDOW_OBJECT *object, UI_EVENT &, EVENT_TYPE ccode)
{
    if (ccode != L_SELECT)
        return (0);
    UIW_POP_UP_MENU *menu = new UIW_POP_UP_MENU(object->trueRegion.left, object->trueRegion.top, WNF_NO_
    FLAGS,
        WOAF_TEMPORARY);
    menu->woStatus = WOS_GRAPHICS;
    *menu
        + new UIW_POP_UP_ITEM("Cancel 2", MNIF_SEND_MESSAGE, BTF_NO_3D, WOF_NO_FLAGS, 0,
            S_CLOSE_TEMPORARY)
        + new UIW_POP_UP_ITEM()
        + new UIW_POP_UP_ITEM("Exit 2", MNIF_SEND_MESSAGE, BTF_NO_3D, WOF_NO_FLAGS, 0,
            L_EXIT_FUNCTION)
        + new UIW_POP_UP_ITEM("Close 2", MNIF_NO_FLAGS, BTF_NO_3D,
            WOF_NO_FLAGS, CloseWindow)

    menu->woAdvancedFlags |= WOAF_TEMPORARY;
    *object->windowManager
        + menu;
    return (0);
}
```

To summarize, to use a pop-up menu without attaching it to a menu bar, create a UIW_POP_UP_MENU object and add it directly to the Window Manager.

6.17 Upgrading applications to run on OpenZinc 1.0 ZD-TN3018

Keywords: Zinc 1.0, Zinc 2.0, Zinc 4.0, upgrade, modify
 Versions: All
 Components: Library, Designer
 Platforms: All
 Issued: 15 May 1995

Question

How do I upgrade my application from Zinc 1.0 or 2.0 to OpenZinc 1.0?

Answer

To upgrade to 1.0 from Zinc 1.0 or 2.0, change the following:

1. Generic()

Search for UIW_WINDOW::GENERIC() and replace it with UIW_WINDOW::Generic() and modify the parameters to match the new form for the generic function. Replace all instances of UIW_SYSTEM_BUTTON::GENERIC() with UIW_SYSTEM_BUTTON::-Generic().

```
UIW_WINDOW::GENERIC(int left, int top, int width, int height, USHORT
woFlags, USHORT
    woAdvancedFlags, int helpContext, char *title);
```

has been changed to:

```
UIW_WINDOW::Generic(int left, int top, int width, int height, ZIL_ICHAR
•title, UI_WINDOW_OBJECT
    *minObject = ZIL_NULLP(UI_WINDOW_OBJECT), WOF_FLAGS woFlags = WOF_NO_
FLAGS, WOAF_FLAGS
    woAdvancedFlags = WOAF_NO_FLAGS, UI_HELP_CONTEXT helpContext = NO_
HELP_CONTEXT) ;

UIW_SYSTEM_BUTTON::GENERIC()
```

has also changed to:

```
UIW_SYSTEM_BUTTON::Generic().
```

Note the change in case in the spelling of Generic, the change in the order of the parameters, the new icon parameter, and the default arguments.

2. windowList

Change all references of "windowList" to "UI_LIST::". UI_EVENT_MANAGER, UI_WINDOW_MANAGER, UIW_WINDOW, and all objects derived from window no longer contain the variable "windowList" because of the multiple inheritance from UI_LIST.

3. .first, last, .current, .previous, and .next

Use First(), Last(), Current(), Previous(), and Next() respectively instead of the member variables *first, *last, *current, *previous, and *next. Those member variables are now protected members of UI_LIST and UI_ELEMENT.

4. Keyboard rawCodes

When using any of the constants for keyboard rawCodes (e.g., ALT_A, F3) place "#define USE_RAW_KEYS" before the "#include <ui_win.hpp>".

5. Palette map structure

Modify any references to the UI_PALETTE_MAP structure to conform to the new structure.

The structure for palette maps has changed slightly. The attrib macro should no longer be called for the graphics mappings. For example:

```
{ ID_WINDOW_OBJECT, PM_ANY, { '   attrib(BLACK, LIGHTGRAY), attrib(MONO_
NORMAL, MONO_BLACK), SOLID_FILL, attrib(BLACK, WHITE), attrib(BW_BLACK,
BW_WHITE), attrib(GS_BLACK, GS_WHITE)} }
```

would become:

```
{ ID_WINDOW_OBJECT, PM_ANY, { '   attrib(BLACK, LIGHTGRAY), attrib(MONO_
NORMAL, MONO_BLACK) , PTN_SOLID_FILL, BLACK, WHITE, BW_BLACK, BW_WHITE, GS_
BLACK, GS_WHITE} }
```

6. Event devices

Move logic for derived devices from the Event() function of those devices to the Poll() function.

The event function of devices attached to the event manager is not called in UI_EVENT_MANAGER::Get function to notify devices of the messages received. For SPY-type devices check the event queue in the Poll() routine using:

```
ccode = eventManager->Get(event, Q_NO_BLOCK I Q_NO_POLL I Q_NO_DESTROY);
```

In this example, ccode will be 0 if there is an event and negative if there isn't. (See UI_EVENT_MANAGER::Get in the OpenZinc Interface Library Programmer's Reference Volume 1.) Also notice that SOLID_FILL has been changed to PTN_SOLID_FILL.

7. UI_DISPLAY

Check calls to display methods to make sure that they follow the new parameter lists.

The functionality of the UI_DISPLAY class has been expanded. Many of the member functions have a different parameter list or do not exist in their original form. See UI_DISPLAY in the Programmer's Reference Volume 1 for specific information on procedures and functions as they now exist.

8. Help files

Move all help for your application into the .DAT file that your application uses.

Help is now stored in the .DAT files rather than in .HLP files. Also the method for converting is now to chose Context I Import from the Help Editor window. All help files will need to be generated from their .TXT files using the import utility from within the designer.

9. Icon format

The bitmap format for UIW_ICON's has changed. Read the chapter on UIW_ICON in the Programmer's Reference Volume 2 for information on the new format.

10. Country information

Change all references to the UI_COUNTRY_INFO structure to appropriate calls to the ZIL_INTERNATIONAL class.

The structure ui_country_info has been replaced by the class ZIL_INTERNATIONAL. The new class will allow for greater flexibility and robustness for handling country-specific information.

11. .DAT file

Convert any .DAT files that were created before OpenZinc 3.0 to the new format using the convert utility available from OpenZinc.

The .DAT file structure has changed significantly. Existing Zinc 2.0 .DAT files can be converted to version 3.0 format using the convert utility `C O N V E R T . Z I P`. After doing this they can be loaded into the OpenZinc 1.0 designer and converted to a format useful to OpenZinc 1.0. More objects can now be stored in the data files and can be accessed within a quicker access time. A data file can contain up to 16,000 objects with the file size being up to 16 megabytes. A single object has a maximum size of 4 megabytes.

12. Device

Change all references to devices to use the new naming format.

The names of the devices have been changed, as follows:

UI_CURSOR	->	UID_CURSOR
UI_BIOS_KEYBOARD	->	UID_KEYBOARD
UI_MS_MOUSE	->	UID_MOUSE

Upgrading applications to run on OpenZinc 1.0 ZD-TN3018

This change was made to provide a better relation between the device classes. All device classes will have the UID_ prefix.

13. Display

Change all references to display classes to conform to the new naming format.

The names of the displays have been changed, as follows:

UI_DOS_BGI_DISPLAY	->	UI_BGI_DISPLAY
UI_DOS_FG_DISPLAY	->	UI_FG_DISPLAY
UI_DOS_TEXT_DISPLAY	->	UI_TEXT_DISPLAY

The _DOS_ portion of the above class names was considered to be redundant. The UI_MSC_DISPLAY class was added to allow display support for Microsoft C/C++.

14. Error system

Replace UI_ERROR_WINDOW_SYSTEM with UI_ERROR_SYSTEM, and UI_ERROR_SYSTEM with UI_ERROR_STUB.

The UI_ERROR_WINDOW_SYSTEM class has been renamed to be the UI_ERROR_SYSTEM class. The old UI_ERROR_SYSTEM class has been renamed to UI_ERROR_STUB. The error system window will display the error text, an "Ok" button, and a "Cancel" button. Selecting the "Ok" button will restore the text that was on the field before the current text was entered. Selecting "Cancel" will keep the text as was entered, regardless of the error message. This allows the error system to function as a dialog window rather than simply reporting the error. This interaction is more standardized with other interface environments.

15. Event manager

Add the same devices to the Event Manager in all environments. The UI_MSWINDOWS_MESSAGE class has been deleted and is no longer required. For an example of how to start an application see the "Getting Started" manual.

16. Help system

Replace references to UI_HELP_WINDOW_SYSTEM with UI_HELP_SYSTEM, and references to UI_HELP_SYSTEM with UI_HELP_STUB.

The UI_HELP_WINDOW_SYSTEM class has been renamed to the UI_HELP_SYSTEM class. The old UI_HELP_SYSTEM class has been renamed to the UI_HELP_STUB class.

17. Hotkeys

Instead of using the hotkey member of the UI_WINDOW_OBJECT class, use the HotKey() function.

UI_WINDOW_OBJECT::hotkey is now private. Use UI_WINDOW_OBJECT::HotKey(). Most member variables should not be changed at random. However, the HotKey() function is provided to allow a means of changing the hotkey.

18. Icons and windows

To use an icon as the minimize icon for a window pass a pointer to it to the constructor of the window.

When an icon is passed in as the minimize object as part of the UIW_WINDOW constructor, it will be displayed when the window is minimized. If the WOS_MINIMIZED flag is set on the window before it is added to the window manager, the window will appear in a minimized state (i.e., as the icon).

19. Jump list

Link in the generated .CPP files that the OpenZinc Designer writes when it saves your .DAT file.

OpenZinc Designer 2.0 created a .CPP file with a UI_JUMP_LIST object that was used to provide a jump to the constructors of the objects created in the designer. OpenZinc Designer 1.0 creates a similar table called _objectTable. _objectTable is an array of UI_ITEM structures. In addition to _objectTable, OpenZinc Designer 1.0 creates _userTable. _userTable is a table of user functions that were assigned to specific objects in the designer. For example, if a button was created (in the designer), and "Function1" was entered for the user function, _userTable will create an entry for "Function1." This change simplifies the process by which objects (created using OpenZinc Designer) can be used in your code. User functions need only be implemented and they will be called automatically.

20. Matrix

Change references to UIW_MATRIX to one of the UIW_HZ_LIST, UIW_VT_LIST, or UIW_TABLE.

The UIW_MATRIX class was divided into the UIW_HZ_LIST (horizontal list) and the UIW_VT_LIST (vertical list). A vertical list will display a single scrollable column of objects. A horizontal list will display multiple columns of objects. The horizontal list is scrollable in the horizontal direction only. For example, if the horizontal list is 5 cell heights tall and 4 cell widths wide, it will have a viewable area of 4 columns containing up to 5 objects each. Any objects added to the lists will be non-editable. These changes were made to provide more standard objects across platforms. This will take advantage of the existing objects (e.g., MS Windows defined objects). Another alternative is to use the UIW_TABLE introduced in OpenZinc 4.0. For examples on how to use the UIW_TABLE see the "Getting Started" manual.

21. Numbers

Use ZIL_BIGNUM and UIW_BIGNUM instead of UIW_NUMBER.

UIW_NUMBER has been replaced by ZIL_BIGNUM and UIW_BIGNUM. This change was to allow greater flexibility regarding numbers. The bignum classes can handle fixed-point numbers that default to 30 digits to the left and 8 digits to the right of the decimal point. The number of allowable digits can be changed by changing the values of ZIL_NUMBER_WHOLE and ZIL_NUMBER_DECIMAL which are #define values located in UI_GEN.HPP. For example, changing ZIL_NUMBER_WHOLE to a value of 60 will

Upgrading applications to run on OpenZinc 1.0 ZD-TN3018

allow the existing code to support 60 digits to the left of the decimal point. ZIL_BIGNUM supports the following operator overloads: =, +, ++, --, +=, -=, ==, !=, >, >=, <=, <. The additional operators will allow greater flexibility in manipulating numerical values. Number formatting (currency, percent, etc.) is supported by the UIW_BKJNUM class only. The UIW_NTEGER class is used to handle integer values only. The UIW_REAL class handles double values and scientific notation. Scientific notation is only used if the length of the number is longer than the length of the visible UTW_REAL field. Limiting the number formatting to the UIW_BIGNUM class reduces the size of the library and the amount of code duplicated in the other number classes. This allows for smaller executable programs.

22. Path

If you are instantiating your own copy of UI_PATH use the new constructor and methods for the UI_PATH class.

The UI_PATH class has been changed. It now implements a linked-list of UI_PATH_ELEMENT objects. The actual path information is contained in UI_PATH_ELEMENT. This change allows new paths to be, more easily, added and deleted from the complete search path (i.e., UI_PATH).

23. Redisplay

Call the Event() function with an S_REDISPLAY to redisplay an object.

In version 2.0, window objects were redisplayed by calling DataSet(NULL). Now objects should be redisplayed by calling their Event() function with an S_REDISPLAY message. Since NULL is equivalent to 0, doing a DataSet(0) will change the number field to 0 rather than re-displaying the field.

24. Scroll bars

Add scroll bars directly to the lists that they are to control.

UIW_SCROLL_BAR objects should be added directly to the object that they will control. In version 2.0, if a scroll bar was added to a parent window directly before a list box, it would control the list box. In version 1.0, the scroll bar should be added to the list rather than before it. This allows the controlling object to have better communication with the scrollbar object.

25. Status flags

Change references to WOAS_* flags to the appropriate WOS_* flags.

The window object advanced status (WOAS_) flags have been changed to window object status (WOS_) flags in order to consolidate the status flags. There were so few of these flags that they were consolidated.

26. Time

Use the new functionality of the ZIL_TIME class.

The ZILTIME class is now derived from ZIL.INTERNATIONAL. This will aid with the conversion between country- specific time formats. The =, +, -, ++, --, >=, <=, !=, +=, -= operator overloads have been added. The additional operators will allow for more ways to manipulate times.

27. User functions

Modify user functions to use the new format.

In version 2.0, user functions were of the following format:

```
void userFunction(void *object, int ccode);
```

In version 1.0, user functions use the following format:

```
EVENT_TYPE userFunction(UI_WINDOW_OBJECT *object, UI_EVENT &event, EVENT_
TYPE ccode);
```

The user function declaration was changed to allow user functions to receive the actual event that caused the function to be called. In version 1.0, user functions will also be called when the associated window object receives the L_SELECT (i.e., <ENTER> is pressed or the object is clicked with the mouse), S_CURRENT (the field becomes current), and S_NON_CURRENT (the field becomes non-current) messages. This will allow user functions greater flexibility. Additionally, user functions are now used as validation routines (e.g., for strings, dates, times, etc.). NOTE: Since existing user functions will now be called three times (on select, current, non-current), it will be necessary to check the ccode at the beginning of each user function to see why it was called. In order to constrain the user function to only execute the body of its code when an L_SELECT message is sent, the following check should be made:

```
if (ccode != L_SELECT)
    return (ccode);
```

28. ui_parse_range

Derive from UIW_STRING or a class derived from UIW_STRING in order to use the ParseRange() function. The parse range function became a protected member of UIW_STRING. This change was made since range parsing is only handled by UIW_STRING and objects derived from UIW_STRING.

29. Validate functions

If you provide a user function for an object, and you need the Validate() function called for that object, then call the Validate function directly.

User functions are now used to validate input fields. The validate functions are no longer used. Please see the "User function" section for details. Having a single function for both purposes eliminates some of the confusion as to the purpose and usage of each function. There is a new function Validate(), that was added to UIW_BIGNUM, UIW_DATE, and UIW_ITIME. This provides the programmer with a pre-defined validation routine. If the userFunction variable for the particular object is NULL, Validate() will be called to validate the value when the object receives the S_CURRENT, S_NON_CURRENT, or L_SELECT messages.

Upgrading applications to run on OpenZinc 1.0 ZD-TN3018

Validate() will call UI_ERROR_SYSTEM::ReportError() to report a validation error. If a user function is provided, Validate() is not called to allow the programmer to either do specialized validation or call the object's Validate() function directly from within the provided user function.

30. UI_STORAGE and UI_STORAGE_OBJECT

Replace UI_STORAGE and UI_STORAGE_OBJECT with the new classes.

The class UI_STORAGE_ELEMENT had been replaced by the class UI_STORAGE_OBJECT. The UI_STORAGE class has been split up and renamed. It now consists of the following classes:

```
ZIL_STORAGE_READ_ONLY
```

```
ZIL_STORAGE
```

The UI_STORAGE_OBJECT class has been split up and renamed. It now consists of the following classes:

```
ZIL_STORAGE_OBJECT_READ_ONLY
```

```
ZIL_STORAGE_OBJECT
```

If no storage functionality is needed or if only read capability is required, only the necessary storage code need be linked in. The inclusion of read or write code is controlled by the ZIL_LOAD and ZIL_STORE macros at compile time.

31. Range strings

Change all range strings to use the newformat.

The format for specifying a range for the UIW_DATE, UIW_TIME, and UIW_BIGNUM objects has changed. A date range must be specified using the following format:

```
YYYY-MM-DD
```

where YYYY is the full year, MM is the month and DD is the day of the month. A time range must be specified using the following format:

```
HH:MM:SS.TT
```

where HH is the hour in 24 hour notation, MM is the minutes, SS is the seconds, and TT are the hundredths (seconds and hundredths are optional). A bignum range must be specified using the following format:

```
[ - ][ nnn ]. [ nnn ]
```

where negative numbers are marked with a minus sign, the decimal separator is a period, and no currency sign is used. Along with the change in how ranges are specified, objects read from a persistent object file are assumed to be in the new format if the file has been saved using the 1.0 storage functions (or created with the 1.0 Designer). If the file is still a 3.6 file (look at the first few characters in the file to see if it is not a 4.x file) then it will be read assuming the range format to be a U.S. format if the ZIL_3x_COMPAT macro was defined when the library was built. This macro is defined by default in UI_ENV.HPP to provide a smoother

upgrade path. Range formats should be updated as soon as possible. You should use the Designer to update your objects, then run GC on the file to convert the file version. After you have done this, you may undefine ZIL_3x_COMPAT and rebuild the library to remove the excess code.

32. New names

Change all old names to use the new names.

Many names have changed in the library. Among these are the following:

<i>Old name</i>	<i>New name</i>
UI_BIGNUM	ZIL_BIGNUM
UI_DATE	ZIL_DATE
UI_TIME	ZIL_TIME
UI_STORAGE	ZIL_STORAGE and ZIL_STORAGE_READ_ONLY
UI_INTERNATIONAL	ZIL_INTERNATIONAL
ichar_t	ZIL_ICHAR
ibignum	ZIL_IBIGNUM
rbignum	ZIL_RBIGNUM
information requests	now have an I_ prefix
ZIL_OLDDEFS	ZIL_OLD_DEFS
SCREENID	ZIL_SCREENID
HBITMAP	ZIL_BITMAP_HANDLE
HICON	ZIL_ICON_HANDLE
IGNORE_UNDERSCORE	FNT_IGNORE_UNDERSCORE
NULLP	ZIL_NULLP
NULLF	ZIL_NULLF
NULLH	ZIL_NULLH
VOIDF	ZIL_VOIDF
VOIDP	ZIL_VOIDP
int8	ZIL_INT8
int16	ZIL_INT16

Upgrading applications to run on OpenZinc 1.0 ZD-TN3018

int32	ZIL_INT32
uint8	ZIL_UINT8
uint16	ZIL_UINT16
uint32	ZIL_UINT32
Max	MaxValue
Min	MinValue
Abs	AbsValue

See the sections of the .HPP files defined for ZIL_OLD_DEFS for a more complete list. Most changes should not affect an application if ZIL_OLD_DEFS is defined when compiling the application. We recommend, however, that applications be fully ported as soon as possible. Most changes were made to avoid the possibility of namespace collisions with other libraries.

33. Internationalization

Use the new internationalization classes.

Many member functions and member variables pertaining to internationalization have been moved from individual classes to the ZIL_LOCALE_MANAGER, ZIL_LOCALE, ZDL_LOCALE_ELEMENT, ZIL_LANGUAGE_MANAGER, ZEL.LANGUAGE, ZIL_LANGUAGE_ELEMENT, ZIL_DECORATION_MANAGER, ZIL_DECORATION, ZIL_BITMAP_ELEMENT, ZIL_TEXT_ELEMENT, ZIL_I18N_MANAGER, and ZILJ18N classes. There are now global instances of the ZIL_LOCALE_MANAGER class (called localeManager), the ZIL_LANGUAGE_MANAGER class (called languageManager), and the Z3L_DECORATION_MANAGER class (called decorationManager). Individual instances of an object can now more easily use different locale and language information.

34. MS Windows screenID

If you are using native windows API calls for a window, make sure that you use the appropriate handle.

In MS Windows a UIW_WINDOW object now has a "client window." A program written at the OpenZinc level will not be affected, but if the application is taking advantage of some Windows API functionality it may need to be adjusted. The OpenZinc parent of an object added to the window will still be the window. The MS Windows parent will be the client window, though. Some objects, such as a status bar or tool bar on an MDI parent window, need to be displayed outside the user-region (client-region) of the window. This change makes this possible.

35. UNICODE.DAT

Make sure the I18N.DAT file is in your path for **Unicode** applications.

The UNICODE.DAT, LANGUAGE.<ISO> and LOCALE.<ISO> data files have been replaced by the I18N.DAT file. This file must be in the path at run-time if internationalization, including character mapping, locale formatting and language translation, is to occur. The global instances of these classes will be created before main starts.

36. UI_MOTIF_DISPLAY

Replaces references to the UI_MOTIF_DISPLAY class with UI_XT_DISPLAY.

The UI_MOTIF_DISPLAY class introduced in OpenZinc 3.5 has been renamed to UI_XT_DISPLAY. The name is more appropriate and will allow easier support for other X-based windowing systems.

37. S_CHANGED

An S_CHANGED event has been added to the library. This event is sent when an object's size or position has changed. When an object receives this event it should update its region as necessary. This event will often be sent in cases where an S_SIZE event used to be sent, so you should trap for S_CHANGED wherever you previously trapped for S_SIZE.

6.18 Deriving a thermometer-type object ZD-TN4005

Keywords: bar chart
Versions: 3.5 and later
Components: Library
Platforms: All
Issued: November 8, 1994
File: **Thermo.zip**

Question

How to create an object that behaves like a thermometer, such as a bar chart-type object that sizes at run time.

Answer

This can be accomplished by deriving your own class from OpenZinc's **UI_WINDOW_OBJECT** class and creating your own **DrawItem()** function and your own **Event()** function.

The **Event()** function receives a size event, modifying the size parameters of the bar chart object and issues a call to the **DrawItem()** function, which redraws the object with the new parameters.

6.19 Using the dot matrix printer

ZD-TN4026

Keywords: UI_PRINTER; dot matrix printer
 Versions: OpenZinc 1.0+
 Components: UI_PRINTER
 Platforms: DOS
 Issued: April 11, 1995

Question

How do I use the dot matrix printer under DOS?

Answer

Under DOS, OpenZinc 1.0 now supports text output to the dot matrix printer. Previously, dot matrix printer support was only available on non-DOS platforms. The functions to accomplish this are `Text()` and `TextFormat()`.

Before using `Text()` or `TextFormat()`, the `UI_PRINTER` instance must be instantiated, and the `BeginPrintJob()` and `BeginPage()` functions must be called. Printing to a dot matrix printer can be specified by setting the `OpenZinc_PRINTER` environment variable to `DM9` or `DM24`, or by passing `PRM_DOTMATRIX9` or `PRM_DOTMATRIX24` as the first parameter to the `BeginPrintJob()` function. `PRM_DOTMATRIX9` and `PRM_DOTMATRIX24` are interchangeable when printing text to a dot matrix printer under DOS.

The `Text()` function prints a word or a line to a specified line and column on the printer page. All `Text()` output must be done from left to right, top to bottom. In other words, you can't output text to the end of a page and then go back to print text at the top.

The `TextFormat()` function prints multiple lines of text. `TextFormat()` word wraps the string automatically and allows you to specify the number of columns to indent the paragraph as well as the starting line for the text.

There are three variables that can be used to adjust output to a dot matrix printer. These are currently private in the `UI_PRINTER` class and should be public in a future release. To modify these for your application, move them from private to public in `ui_dsp.hpp`.

The three variables are:

Using the dot matrix printer ZD-TN4026

- `dotMatrixBottomMargin`
Number of lines to leave for spacing at the end of each page. The default is 8.
- `dotMatrixColumnsPerPage`
Number of columns for the page for word wrapping. The default is 80 characters, but specifying 60, for example, would cause word wrapping to start at column 60.
- `dotMatrixRowsPerPage`
Number of rows per page. This determines when the printer form feeds to the next page. The default is 66.

The top margin is the physical position of the top of the paper in the printer.

See *Reference Manual, Vol. 1* for more information on the `UI_PRINTER` class.

6.20 Differences in platforms—Windows

ZD-TN4031

Keywords: Crossplatform issues; porting; Windows
 Versions: OpenZinc 1.0+
 Components: LIBRARY
 Platforms: Windows
 Issued: May 3, 1995

Question

How do I port an application to Windows or Windows NT?

Answer

In general, porting an application to Windows or Windows NT means recompiling your source code with a compiler supporting the target environment.

The following are some differences and customization issues to be aware of.

1. Application Icon. You can specify the icon for the application in the .rc file.

```
minIcon ICON iconPath
```

2. Changing the application name in the task list. Use the following:

```
windowManger->Information(I_SET_TEXT, "new Name");
```

3. If the system button is not on a window, it cannot be moved, sized, etc. Also, if the system button does not have a move option (MNIF_MOVE) for example, the window cannot be moved.

4. Preventing the user from closing down Windows. In exitFunction, return S_CONTINUE. When a user tries to shut down Windows, the exitFunction can prompt the user to really exit. Returning S_CONTINUE from the exitFunction tells Windows not to shut down.

5. Disk reads from text file; outputting carriage return to text object. Under Windows, a carriage return and line feed must be specified by a \r\n combination. When reading text from disk, make sure the file is opened as binary.

6. Buttons in a UIW_VT LIST. When a button is selected, normally it does not change color to reflect that it is selected. To get this effect, set the two-state flag for the button, or make sure the BTF_NO_TOGGLE flag is off.

7. System events. The event.type of any system specific event, such as a mouse click or keyboard press, has event.type equal to E_OS2. eventmessage.message contains the system event type, such as WM_MOUSEMOVE; event.InputType() returns the OpenZinc event type, which could be E_MOUSE, E_KEY, etc.

6.21 Differences in platforms—OS/2 ZD-TN4033

Keywords: Crossplatform issues; porting application; OS/2
Versions: OpenZinc 1.0+
Components: LIBRARY
Platforms: OS/2
Issued: May 3, 1995

Question

How do I port an application to OS/2?

Answer

In general, porting an application to OS/2 means recompiling your source code with a compiler supporting the target environment.

However, there are some differences and customization issues to be aware of.

1. Application Icon. You can specify an icon for the application in the resource (.rc) file by adding the line shown below.

```
DEFAULTICON iconPath.ico
```

2. Drag and Drop. Drag and Drop is done with the mouse button assigned to drag and drop in the OS/2 setup. OS/2 uses the right button by default.

3. Preventing user from closing down the window. In `exitFunction`, return `S_CONTINUE`. When a user tries to shut down Windows, the `exitFunction` can prompt the user to really exit. Returning `S_CONTINUE` from the `exitFunction` tells Windows not to shut down.

4. Disk reads from text file; outputting carriage return to text object. Under OS/2 a carriage return/line feed must be specified by a `\r\n` combination. When reading text from disk, make sure that the file is opened as "binary." This is the standard in all environments.

5. Buttons in a `UIW_VT LIST`. When a button is selected, it normally does not change color to reflect that it is selected. To get this effect, set the 2 state flag for the button (or make sure the `BTF_NO_TOGGLE` flag is off).

6. Notebooks. The notebook tabs adjust their width automatically.

7. Palette Mapping. This only works when changing the background color of a window. You can, however, derive your own objects with their own `DrawItem()` function which draws the object with the palettes you specify.

8. System events. The eventtype of any system-specific event, such as a mouse click or keyboard press, has an event.type equal to E_OS2; event.message.message contains the system event type, for example, WM_MOUSEMOVE, whereas event.InputType() returns the OpenZinc event type, for example, E_MOUSE, E_KEY.
9. OS\2 Designer. When running the Designer, if the editor icons do not show up, modify the OS\2 system setup. In the OS\2 system folder, under System Settings | System | Window, select Minimize window to desktop.

Platform differences—Motif, Macintosh, NEXTSTEP, Curses ZD-TN4034

6.22 Platform differences—Motif, Macintosh, NEXTSTEP, Curses ZD-TN4034

Keywords: Crossplatform issues; porting application; Motif, Macintosh, NEXTSTEP, Curses
Versions: OpenZinc 1.0+
Components: LIBRARY
Platforms: Motif, Macintosh, NEXTSTEP, Curses
Issued: May 3, 1995

Question

How do I port an application to the Motif, Macintosh, NEXTSTEP and Curses platforms?

Answer

There is a text file for each of these platforms describing the specific things to be aware of when developing for these environments. They also cover some compiler issues with these platforms. The files are:

- mac.txt (Apple Macintosh, PowerPC).
- motif.txt (OSF/Motif). Also available: port.txt and motif.white.paper on the FTP site in pub/doc.
- nextstep.txt (NEXTSTEP).
- curses.txt (Unix Curses).

You can find all the text files in `OpenZinc\` .

6.23 MDI application guidelines ZD-TN4035

Keywords: MDI
 Versions: OpenZinc 1.0+
 Components: LIBRARY
 Platforms: ALL
 Issued: 5-03-95

Question

How do I write an MDI application?

Answer

The following are some guidelines for writing MDI applications:

1. The WOAF_MDI_OBJECT flag must be set on both the parent and child windows.
2. The MDI parent should have a menu. It can also have a toolbar and a status bar. If a toolbar is added it should have its WOAF_SUPPORT_OBJECT flag set (default). It should not have other client area objects, such as buttons, strings, etc., added to it.
3. Events put on the queue that are intended for an MDI child should have the S_MDICHILD_EVENT constant added to it.

For example, to close the current MDI child window:

```
eventManager->Put(UI_EVENT(S_CLOSE + S_MDICHILD_EVENT));
```

4. To get a pointer to the current MDI child use the following:

```
currentChild = mdiParent->Last();
```

5. To enable the use of hotkeys on MDI children, use the following:

```
childWindow->HotKey(HOT_KEY_SUB_WINDOW);
```

6.24 **DGROUP exceeding 64K**

ZD-TN4036

Keywords: DGROUP; DOS Compilers
Versions: OpenZinc 3.5+
Components: LIBRARY
Platforms: ALL
Issued: 5-03-95

Question

How do I get around the linker error of "DGROUP exceeds 64k limit"?

Answer

The DGROUP, which is limited to 64k, is used for global data variables, embedded strings, the stack and the heap.

Here are some ways to free up DGROUP space.

1. Make global variables far.

```
char far *stringName;  
int far someArray[50][50];
```

2. Assign embedded strings to a far char. Instead of...

```
FILE *fh = fopen("dataFile.txt", "r+");
```

... use the following:

```
char far file[] = "dataFile";  
char far mode[] = "r+";  
FILE *fh = fopen(file, mode);
```

3. Also, when linking, use the -zH compiler option. (See your compiler manual if you need more info.)
4. The stack and heap size can also be decreased, although if your program requires a larger stack or heap size, this solution is unworkable.
5. Another solution is to use a 32-bit extender, which eliminates the problem altogether.

6.25 Creating vertical lists with buttons, fonts, and centered text ZD-TN5000

Keywords: bitmap children, button, font, ownerdraw, vertical list
 Versions: 3.5 and later
 Components: Library
 Platforms: Windows, OS/2, Motif, Macintosh, NEXTSTEP
 Issued: July 27, 1994

Question

How do I create a vertical list with buttons, where each button uses a different font and has its text centered?

Answer

In non-DOS environments we create native window objects and in most cases, we let the environment draw the objects for us. But in order to create a vertical list with several buttons, where each button uses a different font and has its text centered, we must have OpenZinc draw the buttons for us. The following code sample shows how to create such a list in Windows, OS/2, Motif, Macintosh and NEXTSTEP.

```
// Create a vertical list and four buttons.
UIW_VT_LIST list = new UIW_VT_LIST(2, 1, 20, 4, NULL, WNF_BITMAP_CHILDREN,
WOFJBORDER);
UIW_BUTTON *btn1 = new UIW_BUTTON(0, 0, 20, "Button A", BTF_NO_3D,
WOF_JUSTIFY_CENTER);
UIW_BUTTON *btn2 = new UIW_BUTTON(0, 0, 20, "Button B", BTF_NO_3D,
WOF_JUSTIFY_CENTER);
UIW_BUTTON *btn3 = new UIW_BUTTON(0, 0, 20, "Button C", BTF_NO_3D,
WOF_JUSTIFY_CENTER);
UIW_BUTTON *btn4 = new UIW_BUTTON(0, 0, 20, "Button D", BTF_NO_3D,
WOF_JUSTIFY_CENTER);
// Set the ownerdraw status for each button in order to call its DrawItem().
// These Flags must be set before the buttons are added to the list.
btn1->woStatus |= WOS_OWNERDRAW;
btn2->woStatus |= WOS_JOWNEKDRAW;
btn3->woStatus |= WOS_OWNERDRAW;
btn4->woStatus |= WOS_OWNERDRAW;
// Assign a different font to each button.
btn1->Font(0);
btn2->Font(1);
btn3->Font(2);
btn4->Font(3);
// Add the buttons to the vertical list.
*list
+ btn1
+ btn2
+ btn3
+ btn4;
```


Creating vertical lists with buttons, fonts, and centered text ZD-TN5000

The above code assumes that the display has its font table initialized with the desired fonts. See other tech notes which explain how to assign fonts to the display's font table in each environment.

6.26 Inserting a new object into a vt. or hz. list at run time **ZD-TN5001**

Keywords: list, object, run time
 Version: 3.0 and later
 Component: Example
 Platforms: All
 Issued: July 7, 1994
 File: None

Question

How do you insert a new object into a vertical or horizontal list at runtime?

Answer

Inserting a new object into a vertical or horizontal list at runtime requires the use of `UI_LIST::Add()`. This overloaded function does not do any of the initialization provided by the `UIW_WINDOW::Add()`, so the programmer must provide it. Below is some sample code showing how to insert a new object (e.g., button) into a vertical list, initialize the object, and redisplay the list.

```
// Create a new button.
UIW_BUTTON *button = new UIW_BUTTON(...);

// Get the current object in the vertical list.
UI_WINDOW_OBJECT *current = vtList->Current();

// Insert the button before the current object.
vtList->UI_LIST::Add(current, button);

// Assign the vertical list as the button's parent.
button->parent = vtList;

// Redisplay the vertical list.
vtList->Information(CHANGED_FLAGS, NULL);
```

See the *Programmer's Reference* for more information on `UI_LIST::Add(UI_ELEMENT ""element, UI_ELEMENT *newElement)`.

6.27 Drag and drop in 4.0

ZD-TN5003

Keywords: drag and drop
Versions: 4.0
Components: Library
Platforms: All
Issued: September 9, 1994

Question

How does drag and drop work in OpenZinc 4.0?

Answer

Objects that may be dragged, or that may receive drops, must be indicated by setting flags on the objects. Flags may be set either in code or in OpenZinc Designer. Relevant flags are:

<i>WOAF_MOVE_DRAG_OBJECT</i>	Object may be move-dragged
<i>WOAF_COPY_DRAG_OBJECT</i>	Object may be copy-dragged (both may be selected)
<i>WOAF_ACCEPTS_DROP</i>	Object may accept the drop of another object

Drag messages

On a mouse down-click, the object under the mouse cursor receives an *L_BEGIN_SELECT* (usually <left-mouse-button>), *L_BEGIN_MOVE_DRAG* message (usually <Shift+left-mouse-button>), or *L_BEGIN_COPY_DRAG* (usually <Ctrl+left-mouse-button>). When this message is received, the object performs two operations immediately:

The object sets *windowManager->dragObject* to point to itself;

The object sets the mouse cursor to the proper image by calling

```
eventManager-> DeviceImage(E_MOUSE, DM_*).
```

Mouse cursor images

Standard mouse cursor images are:

<i>DM_CANCEL</i>	Drop not allowed at this location
<i>DM_DRAG_MOVE</i>	Move of single object allowed to this location
<i>DM_DRAG_COPY</i>	Copy of single object allowed to this location
<i>DM_DRAG_MOVE_MULTIPLE</i>	Move of multiple objects allowed to this location
<i>DM_DRAG_COPY_MULTIPLE</i>	Copy of multiple objects allowed to this location

Drop operations

As the user drags the mouse over objects, OpenZinc does the following:

Checks objects under the drag cursor to see if the object is flagged as *WOAF_ACCEPTS_DROP*. If not, the mouse cursor is set to *DM_CANCEL*.

If the object is flagged as *WOAF_ACCEPTS_DROP*, OpenZinc sends the object's **Event()** function one of the following messages (depending on the *windowManager->dragObject* flags):

<i>L_CONTINUE_SELECT</i>	For default (usually left-mouse-button) drags
<i>L_CONTINUE_MOVE_DRAG</i>	For move (usually Shift+left-mouse-button) drags
<i>L_CONTINUE_COPY_DRAG</i>	For copy (usually Ctrl+left-mouse-button) drags

If the object has a user function, the user function is called. The message passed to the user function depends on the flags of the *windowManager->dragObject* and the message received by the object's **Event()** function. It will be one of the following:

<i>S_DRAG_DEFAULT</i>	Informs the user function of a default drag
<i>S_DRAG_MOVE_OBJECT</i>	Informs the user function of a move drag
<i>S_DRAG_COPY_OBJECT</i>	Informs the user function of a copy drag

In the user function, the programmer decides whether to accept a drop request, and informs OpenZinc of the decision by returning one of the following result codes:

<i>0</i>	User processed message, drop allowed
<i>S_ERROR (-1)</i>	User processed message, drop not allowed
<i>S_UNKNOWN (-2)</i>	User did not process message

If the programmer returns *0* or *S_ERROR*, the user function should set the appropriate mouse cursor image (see cursors above). If the programmer returns *S_UNKNOWN*, or if there is no user function, OpenZinc will send the *S_DRAG_** message to the object's **Event()** function, which will choose the appropriate mouse cursor.

When the user up-clicks, OpenZinc sends one of the following messages to the **Event()** function for the object under the cursor:

<i>L_END_SELECT</i>	End default drag
<i>L_END_MOVE_DRAG</i>	End move drag
<i>L_END_COPY_DRAG</i>	End copy drag

If the object under the up-click is flagged as *WOAF_ACCEPTS_DROP*, and if it has a user function, the user function will be called with one of the following messages:

<i>S_DROP_DEFAULT</i>	Object dropped using "default" drop
-----------------------	-------------------------------------

Drag and drop in 4.0 ZD-TN5003

S_DROP_MOVE_OBJECT Object dropped using "move" drop

S_DROP_COPY_OBJECT Object dropped using "copy" drop

Then, the programmer can decide what to do in the target object's user function when the user drops an object there. The user function should return the same values as the drag queries (section 4 above). If the user function returns *S_UNKNOWN*, or if there is no user function, OpenZinc will send the *S_DROP_** message to the object's **Event()** function, which will perform the default action for a drop on the object.

The following is an overview of the MOVE and COPY actions for different types of OpenZinc library objects:

Move operations

List to List	All objects in the source list flagged as <i>WOS_SELECTED</i> are subtracted from the source list and added to the target list.
List Item to List	The source list item is subtracted from its parent list and added to the target list.
Field to List	This operation is not allowed.
List to Field	If there is one item selected in the source list, then that item is removed from the source list, the text from the item replaces the text on the target field and the item is deleted. If there is more than one item selected in the source list, the target field disallows dropping.
List Item to Field	The source list item is removed from its parent list, its text replaces the text in the target field and the item is deleted.
All others	The text from the source object is used to set the text for the target object, then the text for the source object is blanked out

Copy operations

List to List	All objects in the source list flagged as <i>WOS_SELECTED</i> are duplicated and added to the target list.
List Item to List	The source list item is duplicated and added to the target list.
Field to List	The source field is duplicated and added to the target list.
List to Field	If there is one item selected in the source list, then that item's text replaces the target field's text. If there is more than one item selected in the source list, the target field disallows dropping.
List Item to Field	The list item's text replaces the target field's text.
All others	The text from the source object is used to set the text for the target object.

Notes

- Images for the mouse cursors and key/mouse actions for *MOVE*, *COPY*, and *DEFAULT* are environment dependent, but they are consistent with that environment.
- Unless overridden by the programmer, OpenZinc treats *S_DRAG_DEFAULT* the same as *S_DRAG_MOVE_OBJECT*
- Unless overridden by the programmer, OpenZinc treats *S_DROP_DEFAULT* the same as *S_DROP_MOVE_OBJECT*
- For *MOVE* actions, set the *WOAF_MOVE_DRAG_OBJECT* flag on the source and the *WOAF_ACCEPTS_DROP* flag on the target.
- For moves of a list item to a list, set the *WOAF_MOVE_DRAG_OBJECT* flag on the items in the source list, not on the source list itself.
- For *COPY* actions, set the *WOAF_COPY_DRAG_OBJECT* flag on the source and the *WOAF_ACCEPTS_DROP* flag on the target.

Drag and drop in 4.0 ZD-TN5003

- For copies of a list item to a list, set the *WOAF_COPY_DRAG_OBJECT* flag on the items in the source list, not on the source list itself.
- For all others set *WOAF_DRAG_OBJECT* on the source and *WOAF_ACCEPTS_DROP* on the target.

6.28 Enabling table navigation using keyboard and mouse **ZD-TN5004**

Keywords: table interaction
 Versions: 4.0
 Components: Library
 Platforms: All
 Issued: September 21, 1994

Question

How do you enable navigation through a **UIW_TABLE** object using the keyboard and mouse?

Answer

The **UIW_TABLE** class will have an "edit" mode, which can be toggled on and off at run time.

When the edit mode is on, all OpenZinc standard keyboard interaction will be supported within the current **UIW_TABLE_RECORD** object, unless otherwise specified.

Pressing <Tab> will always move the focus to the next field in a record, unless the last field is current. If the last field is current, and if the table has multiple columns, pressing <Tab> will move the focus from the last field in one record to the first field in an adjacent record. In this manner, pressing <Tab> will always cycle horizontally through the fields and records in a table.

Pressing <Shift+Tab> does the opposite of pressing <Tab>.

When the edit mode is off, the arrow keys will always move the focus between the records in the table and the edit mode will remain off.

When the edit mode is on, the arrow keys will move the focus between the records in the table, only if the arrow keys are not used by a child of the record. If the arrow keys do move the focus between records, the edit mode will remain on.

Pressing the left mouse-button, with the mouse cursor over any field within the table, will cause the table to enter edit mode, and will make that field the current field.

Pressing the left mouse-button, with the mouse cursor over any record other than the current record, but not over any field within that record, will cause the record under the mouse cursor to become current and edit mode to be turned off.

Pressing the left mouse-button, with the mouse cursor over the current record, but not over any field within the current record, will have no effect.

Pressing <Enter> will cause validation of the current record (by sending an *LJSELECT* message to the record's **Event()** function and/or user function) and will toggle the edit mode of the table.

Enabling table navigation using keyboard and mouse ZD-TN5004

If a UIW_TEXT object is the current field in the record, and the table is in edit mode, <Ctrl+Enter> will be required to cause validation and to toggle the edit mode of the table.

Pressing <PageUp> or <PageDown> will scroll the table up or down according to the number of visible records.

See the *Programmer's Reference Volume 2* for more information on the UIW_TABLE and UIW_TABLE_RECORD objects.

6.29 Working with UIW_TABLE_RECORD ZD-TN2008

Keywords: UIW_TABLE, *S_SET_DATA*, *S_NON_CURRENT*, *L_SELECT*, user function
 Versions: 4.0
 Components: Library
 Platforms: All
 Issued: February 6, 1995

Question

What are the similarities and differences between a using a user function with a UIW_TABLE_RECORD and deriving my own table record?

Answer

One way using a user function with a UIW_TABLE_RECORD and deriving your own table record are similar is that they must process the same messages. Both must process *S_SET_DATA* to update data in the table record, as well as *S_NON_CURRENT* and/or *L_SELECT* to update the data in memory or in a database.

An advantage of the user function over deriving a table record is the user function is simpler to use and requires less overhead. However, getting pointers to objects in the table record is required before their data can be retrieved or set. To get the pointers, use the **Get()** or the **Information()** functions. (Because these functions use a linear search they are not very fast.)

The following is an example of a user function:

```
EVENT_TYPE CarRecordFunction(UI_WINDOW_OBJECT *object, UI_EVENT &event, EVENT_TYPE
    ccode)

//Type cast event.data to be the proper type of data structure.
    CAR_ENTRY *carEntry = (CAR_ENTRY*)event.data;
//Update data in memory or in the database.
    if((ccode == S_NON_CURRENT) || (ccode == L_SELECT))
    {
//Use the Get function to get a pointer to a field.
        UI_WINDOW_OBJECT *ownerField = object->Get("OWNER_FIELD");
//Retrieve the data from the field and update the memory or database.
        ownerField->Information(I_COPY_TEXT, carEntry->owner);
        UI_WINDOW_OBJECT *losses = object->Get("LOSSES");
        losses->Information(I_GET_VALUE, &carEntry->losses);
        break;
    }
//Set the data into the table record.
    else if(ccode == S_SET_DATA)
    {
        UI_WINDOW_OBJECT *ownerField = object->Get("OWNER_FIELD");
        ownerField->Information(I_SET_TEXT, carEntry->owner);
    }
}
```

Working with UIW_TABLE_RECORD ZD-TN2008

```
        UI_WINDOW_OBJECT *losses = object->Get("LOSSES");
        losses->Information(I_SET_VALUE, &carEntry->losses);
        break;
    }
}
```

Deriving from **UIW_TABLE_RECORD** is a powerful way to incorporate database support into the table record. However, this requires the additional overhead of including a class declaration and persistence functions. The following is an example of a class declaration:

```
class CAR_RECORD: public UIW_TABLE_RECORD
{
public:
    CAR_RECORD(int width, int height, ZIL_ICHAR *entryOwner);
    virtual ~CAR_RECORD(){}
    virtual EVENTJTYPE Event(const UI_EVENT &event);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void) { return (CAR_RECORD::New); }
    static UI_WINDOW_OBJECT *New(const char *name,
        ZIL_STORAGE_READ_ONLY *file = ZIL_NULLP(ZIL_STORAGE_READ_ONLY),
        ZIL_STORAGE_OBJECT_READ_ONLY *object =
        ZIL_NULLP(ZIL_STORAGE_OBJECT_READ_ONLY), UI_ITEM *objectTable =
        ZIL_MJLLP(UI_ITEM), UI_ITEM *userTable = ZIL_NULLP(UI_ITEM))
    { return (new CAR_RECORD(name, file, object, objectTable, userTable)); }
    CAR_RECORD(const char *name, ZIL_STORAGE_READ_ONLY *file =
        ZIL_NULLP(ZIL_STORAGE_READ_ONLY), ZIL_STORAGE_OBJECT_READ_ONLY *object =
        ZIL_NULLP(ZIL_STORAGE_OBJECT_READ_ONLY), UI_ITEM *objectTable =
        ZIL_NULLP(UI_ITEM), UI_ITEM *userTable = ZIL_NULLP(UI_ITEM));
#endif

protected:
    UIW.STRING *ownerField;
    UIW_NTEGER*losses;
};
```

Because all tables require persistence, you must define **ZIL_LOAD** in the library and include persistence functions in your derived table record. These functions include a **NewFunction()**, which returns the address of the static **New()** function, also declared in the class; and a persistence constructor. The following is an example of a persistence constructor:

```
#if defined(ZIL_LOAD)
CAR_RECORD::CAR_RECORD(const char *name, ZIL_STORAGE_READ_ONLY *directory,
    ZIL_STORAGE_OBJECT_READ_ONLY *file, UI_ITEM *objectTable, UI_ITEM *userTable) :
    UIW_TABLE_RECORD(name, directory, file, objectTable, userTable)
{
//Get pointers to members declared in the class.
    ownerField = (UIW_STRING *)Get("OWNERFIELD");
    losses = (UIW_NTEGER*) Get("LOSSES");
}
```

```

}
#endif

```

In a derived class, the events S_SET_DATA, S_NON_CURRENT, and L_SELECT are handled in the event function instead of the user function. Declare members in the class, that point to the child objects of the table record. This eliminates the need to use the Get() or Information() functions each time one of these events is processed. The following is an example of an Event() function for a derived class:

```

EVENT_TYPE CAR_RECORD::Event(const UI_EVENT &event)
{
    //Map the event. Only keyboard and mouse events need to be mapped. Since many events
    //are processed it is good to avoid mapping when possible. If this function was not
    //trapping for L_SELECT it would be possible to use EVENT_TYPE ccode=event.type
    //instead.

    EVENT_TYPE ccode = LogicalEvent(event);
    switch(ccode)
    {
    //Update data in memory or in a database,
    case S_NON_CURRENT:
    case L_SELECT:
    {
        //Pass event to the base class so the base can do any necessary operations.
        UIW_TABLE_RECORD::Event(event);
        CAR_ENTRY *carEntry = (CAR_ENTRY*)data;
    //Retrieve the data from the fields.
        ownerField->Information(I_COPY_TEXT, carEntry->owner);
        losses->Information(I_GET_VALUE, &carEntry->losses);
        break;
    }
    //Update data in the fields on the record.
    case S_SET_DATA:
    {
        UIW_TABLE_RECORD::Event(event);
        CAR_ENTRY *carEntry = (CAR_ENTRY*)data;
        ownerField->Information(I_SET_TEXT, carEntry->owner);
        losses->Information(I_SET_VALUE, &carEntry->losses);
        break;
    }
    //Pass any events not handled to the base class,
    default:
        ccode = UIW_TABLE_RECORD::Event(event);
    }
    return (ccode);
}

```

Displaying MDI children after adding to MDI parent ZD-TN2010

6.30 Displaying MDI children after adding to MDI parent ZD-TN2010

Keywords: MDI, S_REDISPLAY, S_DISPLAY_ACTIVE

Versions: 3.5 or later

Components: library

Platforms: All

Issued: February 6, 1995

Question

How do I display a MDI child after it has been added to a MDI parent?

Answer

To display an MDI child after it has been added to its parent, tell the parent to redisplay itself. In OpenZinc 3.5 and 3.6 you did this by sending an S_REDISPLAY to the parent.

```
UIW_WINDOW *mdiChild = new UIW_WINDOW("test.dat~MDI_CHILD");
*mdiParent
  + mdiChild;
mdiParent->Event(S_REDISPLAY);
```

In OpenZinc 1.0, the entire parent does not need to be redisplayed. You can send the parent an S_DISPLAY_ACTIVE and the region of the child that needs to be displayed. This results in faster performance. Following is an example:

```
UIW_WINDOW *mdiChild = new UIW_WINDOW("test.dat~MDI_CHILD");
*mdiParent
  + mdiChild;
mdiParent->Event(UI_EVENT(S_DISPLAY_ACTIVE, 0, mdiChild->trueRegion));
```

6.31 Graying out a bitmap on a button when it is nonselectable **ZD-TN4016**

Keywords: Toolbar; button, swapping bitmaps
 Version: 3.6, 4.0
 Component: Persistent Storage; Bitmaps
 Platforms: All
 Issued: January 31, 1995
 Obsolete: NA

Question

How can the bitmap on a button to be grayed out when it is made nonselectable?

Answer

To gray out a bitmap on a button when it becomes nonselectable, replace the existing bitmap with another bitmap that has the desired gray scale such that it gives the visual appearance of being nonselectable.

To get a new bitmap from the Designer, create a button in code that has the bitmap "grayed out" bitmap that you want. This button is never added to the window. It is used only to access our new bitmap.

```
// tempButton is a temporary button that only exists in memory.
// It is never added to the screen.
tempButton = new UIW_BUTTON(0, 0, 0, "", 0, 0, 0, 0, "new_bitmap");
```

Then get the width and height of the new bitmap:

```
int width, height;
tempButton->Information(I_GET_BITMAP_WIDTH, &width);
tempButton->Information(I_GET_BITMAP_HEIGHT, &height);
```

Then set the width and height for this new bitmap using the Information() function.

```
button->Information(I_SET_BITMAP_WIDTH, width);
button->Information(I_SET_BITMAP_HEIGHT, height);
```

Get a pointer to the new bitmap:

```
char *newBitmapArray;
tempButton->Information(I_GET_BITMAP_ARRAY, newBitmapArray);
```

Then set *newBitmap* as the bitmap for the button:

```
button->Information(I_SET_BITMAP_ARRAY, newBitmapArray);
```

Since you never add *tempButton* to the window, delete it before exiting the application.

For a complete example on toggling the selectability and bitmap of a button on a toolbar,

6.32 Displaying geometric objects on a window

ZD-TN4017

Keywords: ellipse, rectangle, text
 Version: 3.6,4.0
 Component: DrawItem(), Derived Objects
 Platforms: All
 Issued: January 31, 1995

Question

How can I display geometric objects on a window without creating a user-defined DrawItem() function for that window?

Answer

You can create geometric objects with their own unique display features by creating a geometric object class from the UI_WINDOW_OBJECT base class, and adding instances of this class directly to a window using the Add() function.

We model our constructor after the display->Ellipse() function such that the parameters passed to the constructor are similar to the parameters passed to the Ellipse() function.

Thus our end usage of our ELLIPSE class will be:

```
*window
+ new UIW_ELLIPSE(200, 200, 50, 100, 0, 360, WOF_NO_FLAGS, BLUE, BLUE);
```

Adding this ellipse to the window would put an ellipse on the window at position (200, 200) with respect to the top left of the window. The ellipse would have an x radius of 50 pixels, and a y radius of 100 pixels. Its start angle would be 0 degrees and its ending angle would be 360 degrees. Its drawing palette would be blue. In designing an ellipse class, first create a common base class, UIW_GEOMETRY_OBJECT. This is used not only as a base class for our ellipse, but it also as a base class for the other geometric objects that we create.

The following is the class declarations for the UIW_GEOMETRY_OBJECT and UIW_ELLIPSE classes.

```
// ellipse.hpp - class declaration for an UIW_GEOMETRY_OBJECT, UIW_ELLIPSE
#include <ui_win.hpp>

class UIW_GEOMETRY_OBJECT : public UI_WINDOW_OBJECT
{
public:
    UIW_GEOMETRY_OBJECT(int _left, int _top,int _right, int _bottom,
        WOF_FLAGS _woFlags, ZIL_COLOR _colorForeground,
        ZIL_COLOR _colorBackground, UI_REGION *_clipRegion);
    EVENT_TYPE Event(const UI_EVENT &event);

    void *Information(ZIL_INFO_REQUEST request, void *data,
```

```
        ZIL_OBJECTID objectID = ID_WINDOW_OBJECT);

    UI_PALETTE *palette;
    UI_REGION *clipRegion;
};

class UIW_ELLIPSE : public UIW_GEOMETRY_OBJECT
{
public:
    UIW_ELLIPSE(int _column, int _line, int _startAngle,
                int _endAngle, int _xRadius, int _yRadius, WOF_FLAGS woFlags = WOF_NO_FLAGS,
                ZIL_COLOR colorForeground = BLACK, ZIL_COLOR colorBackground = WHITE,
                int _fill = FALSE, int _xor = FALSE,
                UI_REGION *_clipRegion = ZIL_NULLP(UI_REGION));

    EVENT_TYPE Event(const UI_EVENT &);
    EVENT_TYPE DrawItem(const UI_EVENT    EVENT_TYPE );

    int column,line;
    int startAngle,endAngle;
    int xRadius,yRadius;
    int fill,xor;
};
```

The DrawItem() function gets called whenever the object needs to draw itself on the screen. The display->Ellipse() function gets called from within the DrawItem() function.

6.33 Deriving a window

ZD-TN1013

Keywords: UIW_WINDOW, Designer, derive
Versions: 3.0+
Components: UIW_WINDOW
Platforms: All.

Question

I can't modify the derived name of a window in the Designer. How do I derive a window that was created with the Designer?

Answer

The way to derive a window from Designer is to create the class definition for your window class as you would create any other derived class, then call the persistence constructor for UIW_WINDOW in the base class constructor. The following snippet of code illustrates the simplest method for this.

```
class WINDOW : public UIW_WINDOW
{
public:
    WINDOW();
    virtual ~WINDOW() {}
    virtual EVENT_TYPE Event(const UI_EVENT &event);
};
WINDOW::WINDOW() : UIW_WINDOW("p_main.dat~WINDOW")
{
    // Center the window.
    windowManager->Center(this);
}
```

If you prefer passing in a pointer to default storage, you can do it as well.

```
class WINDOW : public UIW_WINDOW
{
public:
    WINDOW(ZIL_ICHAR *name, ZIL_STORAGE_READ_ONLY *storage);
    virtual ~WINDOW() {}
    virtual EVENT_TYPE Event(const UI_EVENT &event);
};
WINDOW::WINDOW(ZIL_ICHAR *name, ZIL_STORAGE_READ_ONLY *storage) :
    UIW_WINDOW(name, storage)
{
    // Center the window.
    windowManager->Center(this);
}
```

6.34 Interfacing the UIW_TABLE object to a database

ZD-TN2006

Keywords: UIW_TABLE, database
 Versions: 4.0
 Components: Library
 Platforms: All
 Issued: January 19, 1995

Question

How do I interface the UIW_TABLE object to a database?

Answer

Here's how to interface UIW_TABLE to a database.

Set the WOF_NO_ALLOCATE_DATA flag on the table.

Setting the *WOF_NO_ALLOCATE_DATA* flag tells the table that you will be responsible for keeping track of the data and not to allocate any space. Setting this flag also makes several parameters in the UIW_TABLE constructor irrelevant. They are: *recordSize*, *maxRecords*, and *data*. Set these parameters as follows:

```
recordSize = 0,
maxRecords = -1, and
data = ZIL_NULLP(void).
```

Use InsertRecord() and DeleteRecord() to add and delete records from the table.

InsertRecord() and **DeleteRecord()** add and remove records from the table. When the *WOF_NO_ALLOCATE_DATA* flag is set, **InsertRecord()** and **DeleteRecord()** only adjust the number of records in the table and which record is current. Because they have no effect on the data, records should always be added or removed from the database before either of these functions are called. **DataSet()** can be used to add or remove all records at once. This function also requires all records to be added or removed from the database before it is called.

Use the S_SET_DATA message in the attached user function or the derived table record's Event() function to move data from the database to the table.

Data is moved from the database to the table during processing of the *S_SET_DATA*. The *S_SET_DATA* will be generated whenever a record is added to the table, becomes current or needs to be redrawn. This message should be processed in the user function attached to the table record or in the *Event()* function of a derived table record. When this message is received *event.rawCode* contains the based index of the record that needs its data updated. This number can then be used to access the proper database record and the data can be retrieved. Any necessary conversions can be done and the data placed in the proper fields in the table record.

Use the *S_NON_CURRENT* and *L_SELECT* messages in the attached user function or the derived table record's *Event()* function to move data from the table to the database.

Data is moved from the table to the database during processing of the *S_NON_CURRENT* and *L_SELECT* events in much the same way. The user function or the *Event()* function of a derived table record must process the *S_NON_CURRENT* and *L_SELECT* messages. The data in the table record should be updated to the database when either of these messages are received. This will require getting the data from each field on the table record, doing any necessary conversions and then writing the data out to the database. This must also be done any time that a record's data is changed regardless of whether the data is changed by the end user or programmatically.

Consider the data types needed by the database and the table and match them up as appropriate.

When interfacing the table to a database, consider the type of data stored by the database and the type of data needed by the objects in the table. For example, the database may contain a list of employees, their addresses and salaries. The employees' names and addresses could be stored in the database as a string and the salary could be stored as a double. To display the information, the *UIW_TABLE_RECORD* can contain two *UIW_STRINGS* and a *UIW_BIGNUM*. The strings from the database can easily be placed in a *UIW_STRING*. The real however, cannot be directly placed in a *UIW_BIGNUM*. The real must be converted to a *ZLL_BIGNUM* before it can be placed in the *UIW_BIGNUM*. The conversion of the real to a *ZLL_BIGNUM* and the placing of the data in the table record occurs when you use *S_SET_DATA*. When the data is transferred back to the database, you can remove the text from the *UIW_STRINGS* and place it in the database, but you will also need to remove the *ZLL_BIGNUM* data from the *UIW_BIGNUM*, then export it to a double before storing it in the database. Do this with *S_NON_CURRENT* and *L_SELECT*.

6.35 Viewing system resources at run time **ZD-TN4018**

Keywords: Resource monitor; memory monitor
Version: 3.6,4.0
Component: Event Manager; derived device
Platforms: DOS
Issued: January 31, 1995

Question

How can I monitor a system resource such as available memory when a OpenZinc application is running?

Answer

You can monitor a system resource such as available memory by deriving a device that queries the system for the resource status, and displaying this information top, left corner of the screen in a user created resource monitor window.

Query the system resource in the Poll() function of the resource monitor device. The function for getting the amount of memory in use with the Borland compiler is the heapwalk() function. (Similar functions are available with other compilers.)

From within the Poll() function, use the results of the heapwalk function to compute the total amount of available memory. Send this result directly to a display field in the resource monitor window.

Create the resource monitor window in the constructor of our resource monitor device. Then add the resource monitor window directly to the WindowManager from within the constructor.

6.36 User functions that are members of a class

ZD-TN4019

Keywords: userFunction; callback function
Version: 3.6,4.0
Component: Callback function; user function
Platforms: All
Issued: January 31, 1995

Question

How can a user function be assigned to a OpenZinc object and also be a member of a class?

Answer

A user function, also referred to as a callback function, must be a standard C function with the predefined parameter list that the userFunction declaration expects. Any function that is a member of a class by default is not a standard C function, because it contains a hidden member called the "this" pointer that is a pointer to the class of which the function is a member. When this function is prefaced with the keyword "static," it forces the hidden "this" pointer to be removed from the function. However, static functions can only access other static data members of the class. This makes the non static data member inaccessible to the static function. Also, OpenZinc Designer does not allow user functions that are members of a class to be assigned to an object when in the Designer.

A simple way to resolve both of these problems is by assigning a standard C userFunction to an object, and from within this userFunction, call a nonstatic function that is a class member, passing to the function the same arguments passed to the standard C userFunction.

If you have created a window with a button on it, and you had assigned to the button userFunction MyFunction(), do the following:

```
// Create a standard C userFunction
EVENT_TYPE MyFunction(UI_WINDOW_OBJECT *object, UI_EVENT kevent, EVENT_TYPE ccode)
{
    ((MY_WINDOW *)object->parent)->MyFunction1(object, event, ccode);
}
```

Declare the MY_WINDOW class as follows, creating a nonstatic function, MyFunction1(), that is called by the C userFunction assigned to the button.

```
class MY_WINDOW : public UIW_WINDOW
{
public:
    MY_WINDOW();
    ~MY_WINDOW() {};
    // MyFunction1() has complete access to all class members.
    EVENT_TYPE MyFunction1(UI_WINDOW_OBJECT *object,
```

```
        UI_EVENT &event, EVENT_TYPE ccode);  
private:  
    // any variables  
};
```

The code definition for MyFunction1() would then take some action and return ccode.

```
EVENT_TYPE MY_WINDOW::MyFunction1(UI_WINDOW_OBJECT *object,  
    UI_EVENT &event, EVENT_TYPE ccode)  
{  
    // take some action...  
    return ccode;  
}
```

Displaying timed window on application startup ZD-TN4021

6.37 Displaying timed window on application startup ZD-TN4021

Keywords: UID_TIMER; dialog window
Version: 3.6,4.0
Component: UIW_WINDOW; UID_TIMER
Platforms: ALL
Issued: January 31, 1995

Question

How can an introduction window be displayed at application startup that automatically closes after a set number of seconds?

Answer

This can be done by creating a derived window and adding it to the Window Manager. A UID_TIMER device is then created and added to the Event Manager. This timer device would be told to post a timer message on the queue at a given time interval.

In the Event() function of the derived window we would look for the E_TIMER event. When this occurs, we would create an S_ADD_OBJECT event, setting data equal to a new window that we want to appear on the screen after our introduction window is deleted. We put this S_ADD_OBJECT event on the queue.

We then create a user defined close event, setting data equal to the "this" pointer. We then put this event on the queue. When the Window Manager gets the S_ADD_OBJECT event, it adds the window specified by data to the screen. This new window is now the current window on the screen.

Next, the Window Manager gets our user-defined close event and sends it to the Event() function of the current window. Inside the Event() function of the current window, when we see this user defined close event, we subtract from the windowManager the window specified by data, and then delete this window.

This way, we can put a new window on the screen and close the window behind it.

6.38 Using UIW_TABLE with a database**ZD-TN4026**

Keywords: UIW_TABLE; Database
 Versions: OpenZinc 4.0+
 Components: LIBRARY
 Platforms: All
 Issued: 5-11-95

Question

How do I use the UIW_TABLE object with a database package?

Answer

Here's how to create and use a table in an application using the Designer and a third-party database tool.

This example program creates a table with the following design:

Social Security Number	Name	Address
555-55-5555	John Q Smith	1122 Some Street Dr.

The program requires the following structure to read and write information to the database file.

```
struct SDATA
{
    char firstName[11],
      mi[1],
      lastName[12],
      ssn[9],
      streetData[22],
      city[20],
      state[11],
      zip[6],
      grades[30] [7]; // not used
}
```

Database management functions

The program uses the following functions for database management:

```
long StudentViewRecord(SDATA &sdata, int mode = 0);
```

Pass an SDATA variable into this function with a SSN. If it finds a match, the function returns its index in the database. If no match is found, it returns RECORD_NOT_FOUND (-2).

```
int StudentAddRecord(SDATA sdata);
```

Pass this function an SDATA variable with an SSN; this adds it to the database according to its SSN.

```
int StudentModifyRecord(SDATA sdata,int modifySSN);
```


Using UIW_TABLE with a database ZD-TN4026

```
SDATA StudentGetRecord(long recNum);
```

Pass this function an index into the data base, and it returns the associated record.

```
int StudentDeleteRecord(SDATA sdata);
```

Pass this function an SDATA variable with a SSN, and the with a matching SSN gets deleted from the database.

```
void StudentInitOriginal( );
```

Creates a new random student database.

```
void StudentInitExisting();
```

Uses existing student database.

```
void StudentSave();
```

Saves database info to disk.

Using the Designer to build a table object

(TABLE_WINDOW inside p_event2.dat shows the appearance of the window.)

1. Create a window. Give it the name TABLE in its General page. Add a table object to it. Size the window to the size desired.
2. Size the table. Grab the bottom-right corner of the table with the mouse and size the table to the desired size. You can also size the table in the table's Position page by specifying the desired size. In our example, we size the window to about 75 x 23 cells, almost the entire screen size.
3. Size the table record. Grab the bottom-right corner of the record—the field within the table that looks like a string—with the mouse and size it. (You can size it both vertically and horizontally.) You can also size the table record in the table record's Position page by specifying the desired size.
4. Add fields to the column header. The area that has xxx on it above the table record is the column header. You can drag prompts, strings etc. to the column header. They can then be edited to display the desired text for the column headings of the table. In our example we set the column headings to SSN, Name, Address.

You can also add items to the header from the subobjects notebook page of the header. If a column entry is to be off the screen, the position of the item can be changed from within the position notebook page of the item.

5. Add fields to the table record. From the control panel toolbar, select a formatted string and add it below the SSN column header prompt Set the delete and mask characters so it looks like a SSN field. Give it a name of SSN. (This name is used with the StringID() function to get a pointer to the object.) Add the following seven more strings to the table record. Assign to them the names

<i>Strings</i>	<i>Names</i>
first name	FIRST
middle initial	MI

last name	LAST
street address	STREET_DATA
city	CITY
state	STATE
zip code	ZIP

6. Customize the table parameters. Double-click on the empty area of the table to bring up its notebook editor. On its General page, set columns to 1. (One table record is equivalent to one column. Within the table record you can specify multiple columns as done in step 4.) Set record size equal to the struct used to interface with the data base package. In our example we set this to 71. Also give a unique name to the table. In our example we set Name to TABLE. (This is used with the StringID() function to get a pointer to the table.)

7. Customize the table record. Double click on the table record area. On the Advanced notebook page, assign a callback Function. In our example we use "RecordFunction." This function sets data in the table record from the database and is explained below.

At this point we are done with the Designer aspect of the table program. The following explains what we need to do in code.

Writing code to interface our table object to the database

8. Create the callback function. See RecordFunction() in the event2.cpp module.

9. Create the TABLEJWINDOW class. In the constructor for TABLE_WINDOW, we Get() a pointer to the table and call its DataSet() function. The first parameter is ignored and should be NULL. The second parameter is the number of elements initially used in the table; in our example we set this to 900. The third parameter of the DataSet() function is the maximum number of records. In our example we set this to 1000. You can call the DataSet() function at run time to change these parameters.

10. The Event() function of the TABLE_WINDOW is designed to handle the event processing of the table. It has functionality to

- find a record based on either its index or SSN
- insert a new record
- delete a record based its index or SSN
- delete the record.

Using UIW_TABLE with a database ZD-TN4026

Section 7

OpenZinc and third-party libraries

7.1 MetaWINDOW display class ZD-TN2015

Keywords: graphics display, MetaWINDOW, METAGRAPHICS
 Versions: 4.0+
 Components: graphics library
 Platforms: DOS
 Issued: May 1, 1995

Question

How do I interface the METAGRAPHICS graphics library with OpenZinc?

Answer

If you are using OpenZinc 1.0, you must do two things to interface METAGRAPHICS's MetaWINDOW graphics library.

First, compile D_METDSP.CPP into the library.

You can find D_METDSP.CPP in TN2015.ZIP. To add this to a library, change the library makefile.

Here is an example of how to do this for a 16 bit library in the Borland makefile:

Add the graphics library to the makefile so it will be built.

```
dos16: copy gfx_copy d16_gfx.lib d16_bgi.lib bc_16gfx.lib d16_met.lib test16.exe
```

Specify how to build the d16_met.lib and copy it to the proper lib directory.

```
d16_met.lib : d_metdsp.o16 d_metprn.o16 z_appmet.o16
  -@del d16_met.lib
  $(D16_LIBRARIAN) $(D16_LIB_OPTS)
  $* &
+d_metdsp.o16 +d_metprn.o16+z_appmet.o16
i
  @del zil.sym
  -@md ..\lib\$(VERSION)
  copy d16_met.lib ..\lib\$(VERSION)
```

Specify how to build Z_APP.CPP and D_PRINT.CPP for the METAGRAPHICS graphics library.

```
z_appmet.o16: z_app.cpp $(D16_CPP) -DMETA $(D16_CPP_OPTS) -o$@ $?
```

```
d_metprn.o16: d_print.cpp $(D16_CPP) -DMETA $(D16_CPP_OPTS) -o$@ $?
```

Specify which graphics library to use at link time.

MetaWINDOW display class ZD-TN2015

```
# ---- Use the next line for UI_META_DISPLAY
D16_LIBS=phapi d16_zil d16_met met_xp2d graph286 emu286 emu math1 bc1286
```

Next, place the font files in your path.

Included in the file TN2015.ZIP are the OpenZinc fonts converted to the METAGRAPHICS .FNT file format. Make sure that these files are in your path before the font files for Borland. The Borland font files and the METAGRAPHICS fonts files have the same name. If the Borland fonts precede the METAGRAPHICS fonts in your path the METAGRAPHICS display class will try and load them first and your system will crash. If you wish to use fonts supplied by METAGRAPHICS load the following structure and set it into the font table.

```
METAFont newMetaFont;//Structure definition
newMetaFont.fontPtr = ZIL_NULLP(char);//font pointer
newMetaFont.fontName = "ROMANSIM.FNT";//Font file name
newMetaFont.size =2830;//Size of font file
newMetaFont.maxWidth =8;//Max width of the loaded font.
newMetaFont.maxHeight = 13;//Max height of the loaded font
UI_META_DISPLAY::fontTable[4] = newMetaFont;// Set font into font table.
```

If you are using OpenZinc 4.0, add D_METDSP.CPP to your library as above. You will also need to add the class declaration to UI_DSP.HPP. The class declaration can be found in TN2015.ZIP.

OpenZinc added two new functions to the library for 1.0 that were not available in 4.0: DestroyBitmapHandle() and DestroyIconHandle(). Comment these functions out of D_METDSP.CPP. Finally, add the constructor of the display class to Z_APP.CPP. Add this around line 169, where the different graphics display constructors for DOS are called.

```
# elif defined(META)
// Code for the Metawindow graphics display
if (!display)
    display = new UI_META_DISPLAY;
```

7.2	General Protection Faults	ZD-TN3008
Keywords:	General Protection Fault	
Versions:	3.5 and later	
Components:	Library	
Platforms:	Windows	
Issued:	September?, 1994	

Question

My program quits with a General Protection Fault. What am I doing wrong?

Answer

General Protection Faults are usually related to the following problems:

Problem

The program was compiled without a **.DEF** file to increase the stack and heap size. This causes a GP Fault because of a heap or stack overflow.

Solution

Copy one of the **.DEF** files from the OpenZinc examples.

Problem

The program was compiled with the Borland 4.0 compiler and exception handling and run-time type information was not turned off. OpenZinc's libraries were made with these options turned off. Applications must be internally consistent or they crash frequently.

Solution

Turn off all exception handling and options for your project related to run-time type information. If you are using the Borland IDE see the technote on using the Borland IDE.

An object was added to more than one list. The problem with this is that it causes one of the lists to become corrupted. Each object contains only one set of pointers (parent, previous, next).

Solution

Don't add an object to more than one list.

Problem

A window or an object was deleted from within a user function called by that object or one of its children. You must not delete the object which originates a function. Otherwise, the function will return to an invalid address and crash your application.

Solution

Rather than delete the calling object directly, instead send an event to do the same thing. For example, to close the current window put an *S_CLOSE* event on the event manager's queue:

```
EVENT_TYPE myFunction(UI_WINDOW_OBJECT *object, UI_EVENT &, EVENT_TYPE ccode)
{
    if (ccode != L_SELECT)
        return (ccode);

    object->eventManager->Put(S_CLOSE);
    return (ccode);
}
```

Or to delete an object, put an event on the Event Manager's queue that will tell the object's parent to delete it.

```
const EVENT_TYPE DELETE_BUTTON = 10009;
EVENT_TYPE myFunction(UI_WINDOW_OBJECT *object, UI_EVENT &, EVENT_TYPE ccode)
{
    if (ccode != L_SELECT)
        return (ccode);

    object->eventManager->Put(DELETE_BUTTON);

    return (ccode);
}

MY_WINDOW::Event(const UI_EVENT &event)
{
    EVENT_TYPE ccode = LogicalEvent(event);

    switch (ccode)
    {
        case DELETE_BUTTON:
            UI_WINDOW_OBJECT "button = Get("BUTTON_STRING_ID");
            "this - button;
            delete button;
        }
        default:
            ccode = UIW_WINDOW::Event(event);
    }
    return (ccode);
}
```

7.3	Eliminating optional library components	ZD-TN3009
Keywords:	internationalization, persistence, storage, Unicode	
Versions:	4.0	
Components:	Library	
Platforms:	All	
Issued:	September 15, 1994	

Question

How do I make the libraries without internationalization, persistence, etc.?

Answer

OpenZinc Application Framework has many components which can be compiled in or out of the library depending on the status of specific **#define** switches. To remove an optional component, comment out the related **#define**. To include a component, make sure the appropriate **#define** is active.

The following switches include or remove load and store capability for each object.

```
// Optimization switches for object persistence.
#define ZIL_LOAD// Object Load() member functions.
#define ZIL_STORE// Object Store() member functions.
#define ZIL_STORAGE_IMAGE// OpenZinc UI_STORAGE class.
```

These switches include or remove internationalization capability.

```
// Switches for the international language versions.
ttdefine ZIL_REARRANGEARGS// Support for argument rearrangement in
    // sprintf() and sscanfQ.
#define ZIL_DO_FILE_I18N// Support for Internationalization
#define ZIL_3x_COMPAT// Compatibility for 3x .dat files.
#define ZIL_DO_OS_I18N// Do possibly broken OS i18n support
```

To compile for Unicode support uncomment the following in UI_ENV.HPP. *Note:* Unicode support requires the Unicode Key and basic internationalization support (above).

```
//#define ZIL_UNICODE// Support for UNICODE
//#define ZIL_DECOMPOSE// Support to decompose compound Unicode
//#define ZIL_HARDWARE// Support for Non-AT MS-DOS machines.
```

The **#define** switches can be found in **OpenZinc\SOURCE\UI_ENV.HPP**. After setting or unsetting the appropriate switches the libraries need to be remade. To remake the libraries change to the **OPENZINC\SOURCE** directory and run make with the appropriate make file for your compiler. For instance if the compiler was Borland 4.0 and the target was DOS you would type:

```
make -f btcpp400 dos
```


7.4 Using PharLap 286 with OpenZinc

ZD-TN4004

Keywords: DOS extender, PharLap 286
 Versions: 3.6 and later
 Components: Library
 Platforms: DOS
 Issued: October 28, 1994
 File: None

Question

How do I use the PharLap 286 DOS extender with OpenZinc?

Answer

To compile and run a program using PharLap 286, use the following procedure:

- Rebuild the OpenZinc libraries:
 - Make sure that the OpenZinc include and library files can be found in your path.
 - Make sure that the PharLap include, library, and bin files can be found in your path.
 - Change directories to OPENZINC\SOURCE.
 - Execute the DOS **16** makefile for your compiler
- Modify your makefile to include information for PharLap 286:
 - To bind PharLap to your executable, add the following to the top of your makefile:

```
D16_BIND=bind286
PHARLAP_RTL=c:\phar286\rtl
D16_LOAD=run286a
```

- After the section of the makefile which builds your executable, add:

```
$(D16_BIND) @&&!
$(PHARLAP_RTL)\$(D16_LOAD)
$*
-dll$(PHARLAP_RTL) \moucalls
$(PHARLAP_RTL)\int33
$(PHARLAP_RTL)\doscalls

@del *.sym
```

Section 8

Unicode and i18n

8.1 Decreasing the shipping size of **Unicode.dat** **ZD-TN4003**

Keywords: unicode.dat
Version: 3.6, 4.0
Component: **UNICODE.DAT**
Platforms: ALL
Issued: October 5, 1994
Obsolete: NA
File: 4003.wp

Question

How do I decrease the size of the my shipping copy of **UNICODE.DAT**?

Answer

When your application is ready to ship, you may use **rrmdir** to remove character map tables, locale information, and language strings that won't be used by your application.

If your application is compiled for Unicode you may remove ISO character map tables using:

```
rrmdir il8n.dat ~ZIL_INTERNATIONAL~ISO
```

If your application is NOT compiled for Unicode you may remove the Unicode character map tables:

```
rrmdir il8n.dat ~ZIL_INTERNATIONAL~UNICODE
```

Here is a brief description of each object that you might remove:

Unicode tables:

Shift-US (Japanese DOS, Windows, OS/2)	~ZIL_INTERNATIONAL~UNICODE~IBM_932
IBM 5550 (Chinese DOS for Taiwan)	~ZIL_INTERNATIONAL~UNICODE~IBM_938
KSC 5601 (Korean DOS, Windows)	~ZIL_INTERNATIONAL~UNICODE~IBM_949
BIG 5 (Chinese DOS, Windows for Taiwan)	~ZIL_INTERNATIONAL~UNICODE~IBM_950
GB 2312 (Chinese DOS, Windows)	~ZIL_INTERNATIONAL~UNICODE~IBM_1381
US CP 437	~ZIL_INTERNATIONAL~UNICODE~IBM_437
Greek DOS	~ZIL_INTERNATIONAL~UNICODE~IBM_737
Latin 1 DOS	~ZIL_INTERNATIONAL~UNICODE~IBM_850
Slavic DOS	~ZIL_INTERNATIONAL~UNICODE~IBM_852
Cyrillic DOS	~ZIL_INTERNATIONAL~UNICODE~IBM_855

Turkish DOS	~ZIL_INTERNAHONAL~UNICODE~IBM_857
Portuguese DOS	~ZIL_INTERNATIONAL~UNICODE~IBM_860
Iceland DOS	~ZIL_INTERNATIONAL~UNICODE~IBM_861
Canadian-French DOS	~ZIL_INTERNATIONAL~UNICODE~IBM_863
Nordic DOS	~ZIL_INTERNATIONAL~UNICODE~IBM_865
Cyrillic DOS	~ZBL_INTERNATIONAL~UNICODE~IBM_866
Greek DOS	~ZIL_INTERNATIONAL~UNICODE~IBM_869
Eastern European Windows	~ZIL_INTERNATIONAL~UNICODE~IBM_1250
Cyrillic Windows	~ZBL_INTERNATIONAL~UNICODE~IBM_1251
ANSI/Latin I Windows	~ZIL_INTERNATIONAL~UNICODE~IBM_1252
Greek Windows	~ZIL_INTERNATIONAL~UNICODE~IBM_1253
Turkish Windows	~ZIL_INTERNATIONAL~UNICODE~IBM_1254
Hebrew Windows	~ZIL_INTERNATTONAL~UNICODE~IBM_1255
Arabic Windows	~ZIL_INTERNATIONAL~UNICODE~IBM_1256
Japanese Solaris	~ZIL_INTERNATIONAL~UNICODE~EUC_JIS
Macintosh	~ZIL_INTERNATIONAL~UNICODE~MACINTOSH
NEXTSTEP	~ZIL_INTERNATIONAL~UNICODE~NeXT

ISO 8859-1 tables (see above descriptions):

~ZIL_INTERNATIONAL~ISO~IBM_437
~ZIL_INTERNATIONAL~ISO~IBM_737
~ZIL_INTERNATIONAL~ISO~IBM_850
~ZIL_INTERNATIONAL~ISO~IBM_852
~ZIL_INTERNATIONAL~ISO~IBM_855
~ZIL_INTERNATIONAL~ISO~IBM_857
~ZIL_INTERNATIONAL~ISO~IBM_860
~ZIL_INTERNATIONAL~ISO~IBM_861
~ZIL_INTERNATIONAL~ISO~IBM_863

Decreasing the shipping size of Unicode.dat ZD-TN4003

~ZIL_INTERNATIONAL~ISO~IBM_865
~ZIL_INTERNATIONAL~ISO~IBM_866
~ZIL_INTERNATIONAL~ISO~IBM_869
~ZIL_INTERNATIONAL~ISO~IBM_1250
~ZIL_INTERNATIONAL~ISO~IBM_1251
~ZIL_INTERNATIONAL~ISO~IBM_1252
~ZIL_INTERNATIONAL~ISO~IBM_1253
~ZIL_INTERNATIONAL~ISO~IBM_1254
~ZIL_INTERNATIONAL~ISO~IBM_1255
~ZIL_INTERNATIONAL~ISO~IBM_1256
~ZIL_INTERNATIONAL~ISO~'ACINTOSH
~ZIL_INTERNATIONAL~ISO~NeXT

Delete the languages you aren't supporting:

Catalan ~ZIL_INTERNATIONAL~LANGUAGE~ca
Danish ~ZIL_INTERNATIONAL~LANGUAGE~da
German ~ZIL_INTERNATIONAL~LANGUAGE~de
English ~ZIL_INTERNATIONAL~LANGUAGE~en
Spanish ~ZIL_INTERNATIONAL~LANGUAGE~es
Finish ~ZIL_INTERNATIONAL~LANGUAGE~fi
French ~ZIL_INTERNATIONAL~LANGUAGE~fr
Italian ~ZIL_INTERNATIONAL~LANGUAGE~it
Dutch ~ZIL_INTERNATIONAL~LANGUAGE~nl
Norwegian ~ZIL_INTERNATIONAL~LANGUAGE~no
Swedish ~ZIL_INTERNATIONAL~LANGUAGE~sv

The following languages are all Unicode. You may delete them if you aren't using the Unicode key.

Greek ~ZIL_INTERNATIONAL~LANGUAGE~el
Japanese ~ZIL_INTERNATIONAL~LANGUAGE~ja

Korean -ZIL INTERNATIONAL~LANGUAGE~ko

Delete the locales you aren't using:

Austria -ZIL INTERNATIONAL~LOCALE~AT
English Canada ~ZIL INTERNATIONAL~LOCALE~CA
French Canada ~ZIL INTERNATIONAL~LOCALE~fr_CA
Germany ~ZIL INTERNATIONAL~LOCALE~DE
Denmark -ZIL INTERNATIONAL~LOCALE~DK
Spain ~ZIL INTERNATIONAL~LOCALE~ES
Finland ~ZIL INTERNATIONAL~LOCALE~FI
France ~ZIL INTERNATIONAL~LOCALE~FR
Great Britain ~ZIL INTERNATIONAL~LOCALE~GB
Greece ~ZIL INTERNATIONAL~LOCALE~GR
Italy ~ZIL INTERNATIONAL~LOCALE~IT
Mexico -ZIL INTERNATIONAL~LOCALE~MX
Netherlands ~ZIL INTERNATIONAL~LOCALE~NL
Norway ~ZIL INTERNATIONAL~LOCALE~NO
Sweden ~ZIL INTERNATIONAL~LOCALE~SE
United States ~ZIL INTERNATIONAL~LOCALE~US

These locales are all Unicode. Delete them if you aren't using the Unicode key.

China ~ZIL INTERNATIONAL~LOCALE~CN
Japan -ZIL INTERNATIONAL~LOCALE~JP
Korea -ZIL INTERNATIONAL~LOCALE~KR
Taiwan ~ZIL INTERNATIONAL~LOCALE~TW

lb create an il8n.dat for Japanese DOS/V, Windows, and for English in the US and Canada (because this is Japanese, the code is compiled for Unicode support):

rrmdir il8n.dat ~ZIL INTERNATIONAL~UNICODE~IBM_938
rrmdir il8n.dat ~ZIL INTERNATIONAL~UNICODE~IBM_949
rrmdir il8n.dat -ZIL INTERNATIONAL~UNICODE~IBM_950

Decreasing the shipping size of Unicode.dat ZD-TN4003

rrmdir il8n.dat	~ZIL_INTERNATIONAL~UNICODE~IBM_1381
rrmdir il8n.dat	~ZIL_INTERNATIONAL~UNICODE~IBM_737
rrmdir il8n.dat	~ZIL_INTERNATIONAL~UNICODE~IBM_852
rrmdir il8n.dat	~ZIL_INTERNATIONAL~UMCODE~IBM_855
rrmdir il8n.dat	~ZBL_INTERNATIONAL~UNICODE~IBM_857
irmdiril8n.dat	~ZIL_INTERNATIONAL~UNICODE~IBM_860
rrmdir il8n.dat	~ZIL_INTERNATIONAL~UNICODE~IBM_861
rrmdir il8n.dat	~ZIL_INTERNATIONAL~UNICODE~IBM_863
rrmdir il8n.dat	~ZBL_INTERNATIONAL~UNICODE~IBM_865
rrmdir il8n.dat	~ZIL_INTERNATIONAL~UNICODE~IBM_866
rrmdir il8n.dat	~ZIL_INTERNATIONAL~UNICODE~IBM_869
rrmdir il8n.dat	~ZIL_INTERNATIONAL~UNICODE~IBM_1250
rrmdir il8n.dat	~ZIL_INTERNATIONAL~UNICODE~IBM_1251
rrmdir il8n.dat	~ZIL_INTERNATIONAL~UNICODE~IBM_1252
rrmdir il8n.dat	~ZIL_INTERNATIONAL~UNICODE~IBM_1253
rrmdir il8n.dat	~ZIL_INTERNATIONAL~UNICODE~IBM_1254
irmdiril8n.dat	~ZIL_INTERNATIONAL~UNICODE~IBM_1255
rrmdir il8n.dat	~ZIL_INTERNATIONAL~UNICODE~IBM_1256
rrmdir il8n.dat	~ZIL_INTERNATIONAL~UNICODE~EUC_JIS
rrmdir il8n.dat	-ZIL_INTERNATIONAL-UNICODE-MACINTOSH
rrmdir il8n.dat	~ZIL_INTERNATIONAL~UNICODE~NeXT
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LANGUAGE~ca
nmdiril8n.dat	~ZIL_INTERNATIONAL~LANGUAGE~da
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LANGUAGE~de
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LANGUAGE~es
rrmdir il8n.dat	~ZIL_EVTERNATIONAL~LANGUAGE~fi
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LANGUAGE~fr

Decreasing the shipping size of Unicode.dat ZD-TN4003

rrmdir il8n.dat	~ZIL_INTERNATIONAL~LANGUAGE~it
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LANGUAGE~nl
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LANGUAGE~no
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LANGUAGE~sv
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~AT
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~fr_CA
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~DE
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~DK
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~ES
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~FI
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~FR
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~GB
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~GR
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~IT
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~MX
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~NL
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~NO
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~SE
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~CN
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~KR
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LOCALE~TW
rrmdir il8n.dat	~ZIL_INTERNATIONAL~LANGUAGE~el

Section 9

Help and error system

9.1 Integrating native Windows help with OpenZinc**ZD-TN3013**

Keywords: Help
 Versions: 3.5 and later
 Components: Help system
 Platforms: Windows
 Issued: February 3, 1995

Question

How do I integrate the native Windows help system with OpenZinc?

Answer

To integrate the native Windows help system with OpenZinc, create a Windows-compatible help file, then derive a new class from **UI_HELP_STUB** to call the Windows help engine. To make this as transparent as possible, use OpenZinc Designer to specify help contexts for each object, then pass the contexts programmatically to the native help engine as hypertext lookups.

1. Design your help contexts in the help editor for each object that needs one. Store the keyword for that help context in the title portion of the help. (It would be easiest if you chose the same keywords for the help contexts as the ones for keywords being used.)
2. Using the Designer, design your windows and add the help contexts to the objects.
3. Create a Windows help file. OpenZinc does not provide tools to assist this process. One way to create a Windows help file is to prepare a document in RTF format and compile it using a Windows help compiler. (For more information on how to do this, see the windows programming guide). Alternatively you may use commercial software for preparing help files, such as robohelp.
4. Compile the Windows help file with a help compiler.
5. Derive a new help system from **UI_HELP_STUB** and provide a virtual **DisplayHelp()** function. In this function, OpenZinc will call **WinHelp**.

The following example will not remove the help window from the screen if the application doesn't have a control window; for example, assign a Windows **screenID** to **windowManager->screenID**.

If the record size is set incorrectly in the **.DAT** file and you use **DataSet()** to initialize the table, then you will get unpredictable results. To determine the size of your data structure, write a simple program that prints out the size of the structure. If you are setting the size of the records on multiple platforms and the size of the structure is different on each platform, then you could use **#ifdefs** to make the size of the structures the same across platforms. (This is only important if you are setting the data's size and aren't using the "don't allocate data" flag.) We are in the process of improving the ability to set the record size, and it will probably not need to be read in from the **.DAT** file.

Integrating native Windows help with OpenZinc ZD-TN3013

The following is an example of how to integrate native Windows help with OpenZinc.

```
class ZIL_EXPORT_CLASS MY_HELP_SYSTEM: public UI_HELP_STUB
{
public:
    MY_HELP_SYSTEM(ZIL_ICHAR *winHelpFile, ZIL_ICHAR *zincDatFile,
        UI_HELP_CONTEXT helpContext = NO_HELP_CONTEXT);
    ~MY_HELP_SYSTEM();
    virtual void DisplayHelp(UIJtfINDOW_MANAGER *windowManager,
        UI_HELP_CONTEXT helpContext = NO_HELP_CONTEXT);
protected:
    ZIL_ICHAR *winHelpFile;
    ZIL_STORAGE_READ_ONLY *storage;
    UI_HELP_CONTEXT defaultHelpContext;
    ZIL_SCREENID screenID;
};

static ZIL_ICHAR ZIL_FARDATA _helpDirectory[]= { '~','U','I','_','H','E','L','P', 0 };
static ZIL_ICHAR ZIL_FARDATA _parentDirectory[]= { '.', '.', 0 };
static ZIL_ICHAR ZIL_FAKDATA _currentDirectory[] = { '.', 0 };

MY_HELP_SYSTEM::MY_HELP_SYSTEM(ZIL_ICHAR *_winHelpFile, ZIL_ICHAR *zincDatFile,
    UI_HELP_CONTEXT _helpContext): UI_HELP_STUB(),
    storage(0), defaultHelpContext(_helpContext), winHelpFile(_winHelpFile),
    screenID(0)
{ // Create the storage for the help class to load the help contexts out of
  // the .DAT file
  if (zincDatFile)
      storage = new ZIL_STORAGE_READ_ONLY(zincDatFile);
}

MY_HELP_SYSTEM::~MY_HELP_SYSTEM()
{
    delete storage;
    if (screenID) // only nonzero if help has been used
        WinHelp(screenID, (LPSTR)winHelpFile, HELP_QUIT, (DWORD)0);
}

void MY_HELP_SYSTEM::DisplayHelp(UI_WINDOW_MANAGER *windowManager,
    UI_HELP_CONTEXT helpContext)
{
    if (windowManager->screenID)
        screenID = windowManager->screenID;
    else
        screenID = windowManager->First()->screenID;

    if (storage)
    {
```

```

storage->ChDir(_helpDirectory);
if (helpContext == NO_HELP_CONTEXT)
    helpContext = defaultHelpContext;
ZIL_ICHAR *helpName = storage->FindFirstID(helpContext);
// If the context was found in the .DAT file then load it otherwise display
// the index from the help file.
if (helpName && strcmp(helpName, _parentDirectory) &&
    strcmp(helpName, _currentDirectory))
{
    ZIL_STORAGE_OBJECT_READ_ONLY hFile(*storage, helpName, helpContext);
    ZIL_ICHAR *text;
    hFile.Load(&text);
    if (text) // If text has a value then call WinHelp for that context.
        WinHelp(screenID, (LPSTR)winHelpFile, HELP_KEY, (DWORD)text);
    else
        WinHelp(screenID, (LPSTR)winHelpFile, HELP_INDEX, (DWORD)0);
    delete text;
}
else
    WinHelp(screenID, (LPSTR)winHelpFile, HELP_INDEX, (DWORD)0);
}
// Display the help index if there was no associated .DAT file to get the
// help contexts from
else
    WinHelp(screenID, (LPSTR)winHelpFile, HELP_INDEX, (DWORD)0);
}
UI_APPLICATION::Main()
{
    UI_WINDOW_OBJECT::helpSystem = new MY_HELP_SYSTEM("calc.hip", "p_test.dat");

    UIW_WINDOW *win = new UIW_WINDOW("p_test.dat~RESOURCE_1");
    UIW_WINDOW *win2 = new UIW_WINDOW("p_test.dat~RESOURCE_2");

    *windowManager
    + win
    + win2

    windowManager->screenID = win->screenID;
    Control();

    delete UI_WINDOW_OBJECT::helpSystem;
    return (0);
}

```

Integrating native Windows help with OpenZinc ZD-TN3013

Index

Symbols

- "black box," the 97
- *New()
 - storage
 - using with UIW_IMAGE 23
- *pathName
 - with UIW_IMAGE 20
- .BMP 120
 - using OpenZinc Designer to import 144
- .CPP files 133
- .DAT 138
 - deriving button from a 136
 - using with UIW_IMAGE
 - overview of 17
- .DAT file 137,140
 - loading derived object from 136
 - multiple 131
- .DEF files
 - compiling without 225
- .HPP files 133
- .ICO
 - using OpenZinc Designer to import 144
- .TXT
 - importing using OpenZinc Designer 144
- .XPM
 - using OpenZinc Designer to import 144
- .Z T 143
- _className
 - used in UIW_IMAGE 25
- JielpDirectory 238
- _serviceManager 149

Numerics

- 3D controls
 - how to use with OpenZinc 51
 - overview of 51

A

- accelerator keys 68
- access
 - defined in ZIL_FILE 30
 - description of 36
- actions
 - COPY 200
 - MOVE 200
- Add()
 - behavior of in UI_LIST 57
 - virtualized 77
- adding 162
- adding fonts 88
- AGENCY.ZIP 160
- ANALOG
 - UIW_DATE used in 68
- ANSI/Latin I 231

Apple menu item

- how to implement 64
- Arabic 231
- argv 55
- argv 55
- ATCF_FLAGS
 - checking with inline functions 42
- Attachments
 - checking with inline functions 42
- Austria 233

B

- Backspace
 - under Macintosh 69
- bar chart
 - creating 186
- BeginPrintJob() 59
- BIG 5 230
- bit masking
 - using to change status of a flag on an object 155
- bitmap children 195
- BITMAP_OBJECT 121
- bitmaps
 - swapping when button is nonselectable 209
 - using with OS/2 120
- Borland 81
 - PowerPak 32 54
- Borland 4.0 compiler 225, 227
- Borland IDE 81
- BTF_FLAGS
 - checking with inline functions 42
- BTF_SEND_MESSAGE 154
- BTS_FLAGS
 - checking with inline functions 42
- button 209
 - graying bitmap on a 209
 - in vertical list 195
 - togglng 156
 - togglng on a window 156
 - using to close window 154
 - checking with inline functions 42
 - in a list with different fonts 195
- button status
 - checking with inline functions 42

C

- callback function 97,216
- Catalan 232
- ccode 205,207
- centering object 60
- CHANGED_J^LAGS 155, 163
- CHANGED_STATUS 155, 165
- check box

- deselecting 156
- selecting 165
- CHR 90
- class
 - initializing information in a 166
- class description
 - of UIW_MAGE 21
- class overview
 - of ZIL_FILE 27
- close window 154
- Close()
 - definition of 29
 - description of 33
- code readability
 - improving 41
- color 117
- columns
 - in UIW_TABLE 71
 - using with the table object 160
- columns in lists 160
- COLUMNS.ZIP 160
- combo box
 - nth element within 161
- CommitStack()
 - allocating stack space using 54
- compiler
 - Borland 4.0 227
- constructor
 - calling the base class 166
 - new in UIW_WINDOW 65
 - persistence 206
 - persistent 166
- UIW_IMAGE 22
- copy operations 201
- CPP 131,133
- CreateWin()
 - called CreateWindow() 68
- CreateWindow() 68
- current
 - forcing an object to remain 168
- Curses 117
- cursor
 - changing to an hourglass 122
- CURVALDD.ZIP 169
- Cyrillic 231

- D
- D_ERRORI.CPP 98
- D_GFXDSP.CP 91
- D_GFXDSP.CPP 90
- Danish 232
- DAT file
 - exporting to text 129

- DAT files 131,133
 - using multiple in one application 131
- database 205
 - third-party
 - using with OpenZinc table object 213
 - using with UIW_TABLE 219
- DataSet() 137,138
 - initializing table with 71
- Dates
 - checking with inline functions 43
- default printing 59
- defaultInitalized
 - in ZAF_MESSAGE_WINDOW 66
- Defaults Editor
 - description of 73
- Delete
 - under Macintosh 69
- Delete Right
 - under Macintosh 69
- Denmark 233
- derive 212
- derived object
 - loading from .DAT 136
- Designer 125, 164
 - using a table created in the 145
- DestroyImageHandle(void)
 - description of 25
- destructor of
 - UIW_IMAGE 22
- dialog font
 - loading into fontTable 91
- dialog window 218
- Dimension constraints
 - checking with inline functions 42
- DIRECT.HPP 125
- Directory services 125
- directory services
 - using Service Manager with 128
- display
 - font table 196
 - using METAGRAPHICS 223
- display mode
 - changing at run time 113
- DisplayHelp(237
- DisplayHelp() 237
- DM_CANCEL 198, 199
 - description of 74
- DM_DRAG 47
- DM_DRAG_COPY 198
 - description of 74
- DM_DRAG_COPY_MULTIPLE 198
 - description of 74
- DM_DRAG_MOVE 198
 - description of 74

- DM_DRAG_MOVE_MULTIPLE 198
 - description of 74
- DNCF_FLAGS
 - checking with inline functions 42
- DOS 98,117, 164,230
 - Borland
 - stack size with 78
 - Canadian-French 231
 - Chinese 230
 - Cyrillic 230
 - Greek 230
 - Iceland 231
 - Latin 230
 - Microsoft
 - stack size with 78
 - Nordic 231
 - Portuguese 231
 - Slavic 230
 - Symantec
 - stack size with 78
 - Taiwan 230
 - Turkish 231
 - using display classes with 77
 - Watcom
 - stack size with 78
- DOS extender 228
 - Pharlap 286 228
- dot matrix printer
 - support for in UI_PRINTER 187
- drag and drop 198
 - in OpenZinc Designer 73
- dragObject 198
- DrawFocus()
 - drawing focus on object with 61
- drawing 110
- DrawItem() 110,117,120, 160,186, 211
 - using with UIW_IMAGE 18, 22
- DTF_FLAGS
 - checking with inline functions 43
- Dutch 232
- E**
- E_DEVICE
 - using with DeviceState() 56
- E_TIMER 218
- Eastern European 231
- ellipse 210
- EndPrintJob(59
- English 232
- English Canada 233
- Epson-compatible dot-matrix printer
 - using with OpenZinc 58
- errata
 - overview of 55
- error 140
 - defined in ZIL_FILE 30
 - description of 36
 - checking 140
- ERROR_ACCESS
 - description of 32
- ERROR_NAME
 - description of 32
- ERROR_NONE
 - description of 32
- ERROR_NULL_STRING
 - description of 32
- ErrorMessage() 98
- event
 - flow 96
 - interpretation 100
 - receiving in a window 168
 - user-defined 168
 - user-defined close 218
- event flow
 - bottom-up 96
 - NEXTSTEP 96,97
 - top-down 96
- EventManager 96,168, 218
 - using to monitor system resource 215
- event map table 101
- event queue 96,168,226
- Event() 100, 102, 126, 162,186, 218
 - selecting mouse cursor images with 199
 - using to redisplay a new item in a list 162
 - using with UIW_IMAGE 18,22
- event.data 137
- event.rawCode 137
- event.text
 - deallocating 51
- EVENT_TYPE 98
- EventManager 149
- eventMapTable 100
- events
 - example of using programmatically 49
 - interpreting 100
 - open and print document 48
- example
 - LSTITM 110
- exception handling 81
 - with the Borland compiler 225
- exitFunction
 - definition of 60
- export
 - using OpenZinc Designer to files 144

F
field
 invalid 168
FILEEDIT 125
Finish 232
Finland 233
fixed-width font 94
flag
 changing at run time 155
flags
 how to use 105
FlagSet() 118
 supplemented with inline functions 41
FlagsSetO 118
 using multiple functions instead of 41
font 89, 90
 changing for object 88
 GFX 90
 Motif 92
 on buttons 195
 OS/2 89
 variable 91
font table 196
 initializing with fonts 196
Font() 88
FontRec
 changed to FontInfo 58
fonts
 changing object 88
fontTable 88, 89, 90
 fixed-width fonts 94
 GFX 90
 OS/2 89
France 233
French 232
French Canada 233

G
g_pnorm.cpp 118
GB 2312 230
General Protection Fault 225
general protection fault 225
geometric objects
 displaying 210
Geometry
 notebook page
 selecting relative constraint in 70
geometry management
 removing 152
German 232
Germany 233
Get(161
Get() 164, 205

GetError(void)
 definition of 28
Getting Started 110
 errata in 67
GFX 90
GFX fonts 90
GpiLoadBitmap 121
Great Britain 233
Greece 233
Greek 231

H
handle
 defined in ZIL_FILE 30
 description of 36
heap size
 minimum size for Macintosh 78
Hebrew 231
Help Editor 144
help system
 integrating native with OpenZinc 237
help system
 overview of 40
HELP_INDEX 239
helpContext 239
horizontal list 161
HPP 131

I
I_ACTIVATE_SERVICE 149
I_GET_BITMAP_HEIGHT 209
I_GET_BITMAP_WIDTH 209
I_GET_VALUE 159
I_INITIALIZE_CLASS 166
I_SET_REQUESTOR 149
IBM 5550 230
ICF_FLAGS
 checking with inline functions 43
icons
 checking with inline functions 43
IDE 81
 Borland 81
 Microsoft 83
 Watcom for Windows 85
image
 located in UIW_IMAGE
 member variable of UIW_IMAGE 25
 mouse cursor 122
Image Editor 144
 grouping objects in 73
imageHeight 25
images
 checking with inline functions 43
imageWidth 25

- IMF_BACKGROUND 18,22
 - description of 21, 22
- IMF_FLAGS 25
 - checking with inline functions 43
 - definition of 18
- IMF_NO_FLAGS 18
 - description of 21
- IMF_SCALED 18,22
 - description of 21
- IMF_TILED 18,22
 - description of 21
- imFlags
 - used in UIW_IMAGE 25
- implementation
 - of UIW_IMAGE 18
 - of ZIL_FILE 27
- import
 - using OpenZinc Designer to 144
- Index
 - using with help system 40
- information requests
 - UIW_TIME 64
- Information) 149, 155,166,205
- Inheritance
 - of UIW_IMAGE 18
- INITIALIZE_CLASS 166
 - becomes I_INITIALIZE_CLASS in 4.0 166
- inline function 41
- internationalization 227
 - eliminating 227
- invalid field
 - forcing to be current 168
- IsBackground(void) 18,23
- ISO
 - character map tables 230
- ISO 8859-1 150
 - tables 231
- ISO_DIAL.CPP 90, 91
- IsScaled(void) 19,23
- IsTiled(void) 23
 - using with UIW_IMAGE 19
- Italian 232
- Italy 233

J

- Japanese 231
 - DOS 230
 - OS/2 230

K

- keyboard
 - mapping
 - gray keys in 69
- KSC 5601 230

L

- L_BEGIN_COPY_DRAG 198
 - description of 75
- L_BEGIN_MARK 102
- L_BEGIN_MOVE_DRAG 198
 - description of 75
- L_BEGIN_SELECT 198
- L_CONTINUE_COPY_DRAG 199
 - description of 75
- L_CONTINUE_MOVE_DRAG 199
 - description of 75
- L_CONTINUE_SELECT 199
- L_END_COPY_DRAG
 - description of 75
- L_END_MOVE_DRAG
 - description of 75
- L_NEXT 96
- L_SELECT 102,203, 205, 207, 214
- language
 - selecting a new 68
- Length()
 - description of 33
- LIB 125
- lifetime
 - of an object 166, 167
- list
 - adding object at run time 197
 - columns 160
 - nth element within 161,162
 - object added to a 166
 - redisplaying 162
 - removing an object from a 166
 - retrieving items from a 161
- Lists
 - checking with inline functions 46
- lists
 - forcing an item to remain current 169
- LoadImageFromApplication()
 - using with pathName 22
- LoadImageFromApplication(void) 20
 - description of 25
- LoadImageFromFile()
 - using with pathName 22
- LoadImageFromFile(void) 20
 - description of 25
- loading derived object from .DAT 136
- local control loop 98
 - implementing 98
- location
 - of ZIL_FILE definition 27
 - UIW_IMAGE definition 18
- logical events
 - those generic to all platforms 53

LOGICAL_PALETTE 118

LogicalEvent() 101

LogicalPalette() 119

loop

control 98

local 98

main 98

modal 98

while 99

LOOP_CANCEL 98, 99

LOOP_OK 99

LOOP_WINDOW 98

LSTITM 110

M

Mac

file and edit menus in 76

Macintosh 164,195,231

Clipboard 76

grouping objects on 70

keyboard mapping under 69

libraries in 73

PowerMacintosh 78

restrictions using drag-and-drop under 49

Macintosh libraries

contents of listed in **Writing Multiplatform**

Programs 67

listed in **Writing Multiplatform Programs** 67

Mac **ZIL** II 68

MAGIC_NUMBER

using with **.DAT** file security 134

makefile

information about **PharLap** 286 228

map table

for events 101

MapPalette() 119

mappings

keyboard and mouse

under **Macintosh** 69

MapRGBColor()

used in **UI_MACINTOSH_DISPLAY** 57

maxColors

used in **UI_MACINTOSH_DISPLAY** 57

MDI

displaying children 208

MDI window 208

displaying 208

Member functions

of **UIW_IMAGE** 18,22

protected 25

member variables

in **ZAF_MESSAGE_WINDOW** 66

UIW_IMAGE

protected 25

memory

monitoring available 215

memory monitor 215

Menu items

checking with inline functions 44

Message Editor 144

message passing 96

MessageBox() 98

messages

during object's lifetime 166

operating environment-specific 97

OpenZinc-specific 97

METAGRAPHICS 223

MetaWINDOW 223

Mexico 233

Microsoft 77

IDE 83

MNIF_ABOUT

using with **Macintosh** 63

MNIF_FLAGS

checking with inline functions 44

mode

defined in **ZIL_HLE** 30

description of 36

Motif 164, 195

DESQview/X

stack size with 78

fonts 92

QNX

stack size with 78

mouse

cursor 122

mouse cursor

changing to drag image 47

setting image of

drag and drop 198

MOVE 201

move operations 201

MY_BTN.ZIP 136

myLanguage

in **ZAF_MESSAGE_WINDOW** 66

N

Netherlands 233

New() 136

NewFunction()

used with **UIW_IMAGE** 23

NEXTSTEP 195,231

file and edit menus in 76

Pasteboard 76

using **UIW_IMAGE** with 17

NMF_DECIMAL

was **NMF_DIGITS** 63

- NMF_DIGITS
 - changed to NMF_DECIMAL 63
- NMF_FLAGS
 - checking with inline functions 44
 - supplanted with inline functions 44
- nonfield region
 - equivalent of in UIW_IMAGE 21
 - equivalent of with UIW_IMAGE 21
- nonstatic function 216
- NormalizePosition()
 - description of 52
- NormalizeString()
 - description of 52
- Norway 233
- Norwegian 232
- nth item 161
- numberID 131
- Numbers
 - checking with inline functions 44
- 0
- O_DSP.CPP 89
- object
 - adding to a list at run time 162
 - adding to list 162,197
 - adding to the Window Manager 166
 - deriving
 - bar chart 186
 - in OpenZinc Designer 141
 - inserting into a list 197
 - messages 166
 - moving at run time 163
 - moving on a window 163
 - pointer to 168
 - reactivating on Macintosh 70
 - reading from a .DAT file
 - errors in 140
- objectID 155
- objects
 - dragging 198
 - dropping 198, 199
 - grouping on Macintosh 70
- objectTable 25
 - using with UIW_IMAGE 24
- Open()
 - definition of 29
 - description of 33
- opening
 - implementing with S_OPEN_DOCUMENT 48
- operating system
 - ISO name of current 62
- operations
 - move 201
- Opt+Tab
 - under Macintosh 69
- OS/2 98,164, 195
 - bitmaps under 120
 - fonts 89
 - stack size with 78
 - using bitmaps with 120
 - using UIW_IMAGE with 17
- OSF/Motif 17
 - using UIW_IMAGE with 17
- ownerdraw 195
- P
- p_stredit.znc 149
- palette mapping 117
- PALETTE.ZIP 117
- palette.zip 119
- paletteMapTable 119
- parent
 - of an object 168
- paste buffer 76
- pathName
 - description of 36
 - defined in ZIL_FILE 30
 - with UIW_IMAGE 22
- pattemTable
 - used in UI_MACINTOSH_DISPLAY 57
- pcx
 - using with UIW_IMAGE 17
- persistence 141,206,227
 - eliminating 227
 - using with UIW_IMAGE 17
 - with UIW_IMAGE 23
- PharLap
 - using with OpenZinc 228
- PharLap 286 228
- pointer 161
- Poll() 215
- portability
 - of UIW_IMAGE 18
 - of ZIL_FILE 27
- POSIX
 - using with ZIL_FILE 36
- PostScript
 - using with UI_PRINTER 60
- Power Macintosh
 - see Macintosh 78
- PowerPak 32
 - crashing applications with
 - fixing 54
- printing
 - implementing with S_PRINT_DOCUMENT 48
 - using dot-matrix printer 58
 - OpenZinc recommendations for behavior concerning 51

PRM_DOTMATRIX24 187
PRM_D0TMATRIX9 187
PRM_LANDSCAPE
 environment variable 60
Programmer's Reference Volume One
 errata in 55
Programmer's Reference Volume Two
 errata in 62
Protected
 members of UIW_IMAGE 20
protected
 members of UIW_IMAGE 25
PTN_INTERLEAVE_FILL
 used on UI_MACINTOSH_DISPLAY 58
PTN_RGB_COLOR
 used on Macintosh 57
Public
 description of UIW_IMAGE members 21
pull-down item
 send flag 71
pull-down menu
 appearing while editing window under Mac 70

Q

QuickDraw
 used in UI_MACINTOSH_DISPLAY 58

R

radio button
 selecting 165
Read()
 defined with ZILJJNICODE 29
 definitions of 29
 description of 34
 using with Unicode 35
record size
 in UIW_TABLE 71
rectangle 210
region
 drawing focus in 61
 with IMF_SCALED 21
 with IMF_TILED 21
Relative constraints
 checking with inline functions 44
relative constraints
 in OpenZinc Designer 70
relative position
 determines screen position 166
Rename()
 definition of 29
 description of 34
resolution
 changing at run time 113

resource files
 overview of using with UIW_IMAGE 17
resource monitor 215
rgbColorMap
 used in UI_MACINTOSH_DISPLAY 57
RLCF_FLAGS
 checking with inline functions 44
RTTI options 81
runtime 155
 adding a list or object at 197
 adding object to list 162,197
 changing flags 155
 changing resolution 113
 changing status 155
 moving object 163
 type information
 with the Borland compiler 225

S

S 162
S_ADD_OBJECT 218
S_CLOSE 96, 103,154, 166, 226
 trapping the 103
S_CREATE 121, 158, 167
S_IDEINITIALIZE 103, 166,167
S_DISPLAY_ACTIVE 208
S_DRAG_COPY_OBJECT 199
 description of 74
S_DRAG_DEFAULT 199,201
 description of 75
S_DRAG_MOVE_OBJECT 199, 201
 description of 74
S_DROP_COPY_OBJECT 200
 description of 75
S_DROP_DEFAULT 199,201
 description of 75
S_DROP_MOVE_OBJECT 200, 201
 description of 75
SJ5RROR 199
 description of 76
S_INITIALIZE 166,167
S_MDICHILD_EVENT 154
S_NON_CURRENT 139, 146,168, 205, 207, 214
S_OPEN_DOCUMENT 48
 how to use 50
S_PRINT_DOCUMENT 48
 how to use 51
 posting to print queue 51
S_REDISPLAY 117,162, 208
S_SET_DATA 139,146, 205, 207, 213
S_SIZE 117
S_UNKNOWN 199
 description of 76
S_VSCROLL 159

- screenID 166
 - Windows 237
- scroll bar
 - default width and height of 64
- ScrollEvent() 158,159
- scrolling
 - UIW_TABLE 158
- security
 - using with .DAT file 134
- SEEK
 - type definition 28
- Seek()
 - definition of 29
 - description of 34
- SEEK_FROM_CURRENT
 - description of 32
 - using with Seek() 34
- SEEK_FROM_END
 - description of 32
 - using with Seek() 34
- SEEK_FROM_START
 - description of 32
 - using with Seek() 34
- selecting a button 165
- Service Manager 128
- service.lib
 - building 148
- services
 - directory 125
- setError()
 - definition of 29
- Shift-JIS 230
- Solaris
 - Japanese 231
- Spain 233
- Spanish 232
- stack
 - allocating space on 54
 - recommend size with OpenZinc 78
- status
 - changing at run time 155
 - changing on a flag or an object 155
- STF_FLAGS
 - checking with inline functions 45
- Storage
 - checking with inline functions 45
- storage
 - eliminating 227
 - limitations of with UIW_IMAGE 17
 - using UIW_WINDOW constructor 65
- STREDIT.HPP 149
- string
 - converting 8-bit to 16-bit 52
- String Editor 148
- STRING_FIELD 126
- stringID 126
- strings
 - checking with inline functions 45
- Subtract()
 - virtualized 77
- swapping bitmaps 209
- Sweden 233
- Swedish 232
- SYF_FLAGS
 - checking with inline function 45
 - supplanted with inline functions 45
- system buttons
 - checking with inline functions 45
- system events
 - those available to the user 53
- T
- table 137, 145, 206
 - displaying info in 145
 - navigating 203
 - record in a 205
- table interaction 203
- table record
 - number 138
- TableRecordCallBack() 137
- Tell()
 - description of 33
- TEST1.CPP 132
- TEST2.CPP 132
- text 210
 - centered 195
 - changing font 88
 - exporting from .DAT file 129
- text field
 - processing raw events with 100
- TextFormat()
 - in UI_PRINTER 60
- thermo.zip 186
- TIFF images
 - using with UIW_IMAGE 17
- time
 - checking with inline functions 45
- timer 218
- title 164
 - subtracting from a window 164
- TMF_FLAGS
 - checking with inline functions 45
- toggle buttons 156
- toolbar 209
- TrueToRelative()
 - description of 52
- Turkish 231
- types and constants
 - of UIW_IMAGE 21

u

- UI_APPLICATION
 - argc and argv 55
- UI_CONSTRAINT 152
- UI_DISPLAY
 - Text 56
 - left and right in 56
 - using VirtualGet() and VirtualPut() with 56
- UI_ELEMENT 197
- UI_EVENT_MANAGER
 - using DeviceState() 56
- UI_GEOMETRY_MANAGER 152
 - pixel and minicell coordinates in 56
- UI_GRAPHICS_DISPLAY 88
- UI_HELP_CONTEXT 238
- UI_HELP_STUB 237
- UI_HELP_SYSTEM
 - SetLanguage 57
- UI_LIST
 - Add() 57
 - Add() and Subtract() in 77
- UI_MACINTOSH_DISPLAY
 - FontRec 58
 - MapRGBColor() 57
 - maxColors 57
 - patternTable 57
 - PTN_INTERLEAVE_FILL 58
 - PTN_RGB_COLOR 57
 - rgbColorMap 57
- UI_MSC_DISPLAY
 - using with Microsoft compiler 77
- UI_PALETTE_MAP 119
- UI_PRINTER
 - BeginPageO 59
 - BeginPrintJob() 59
 - dot-matrix printer support 58,187
 - EndPrintJob() 59
 - how to print documents with 58
 - using UI_PRINTER 58
 - using with S_PRINT_DOCUMENT 51
- UI_REGION 110,211
- UI_RELATIVE_CONSTRAINT 60
- UI_SCROLL_INFORMATION 159
- UI_STORAGE_OBJECT_READ_ONLY
 - see ZIL_STORAGE_OBJECT_READ_ONLY 69
- UI_WCC_DISPLAY
 - using with Watcom compiler 77
- UI_WINDOW_MANAGER 103
 - adding print functionality to 51
 - misspelled as UI_WINDOW_MANGER 67
- UI_WINDOW_OBJECT 88, 96,100,110,117,186
 - DrawFocus() 61
 - helpSystem 239
- UID_TIMER 218
- UIF_FLAGS
 - type definition of 28
 - using in place of UIF_STATUS 155
- UIS_BINARY 28
 - description of 31
- UIS_COPY 28
 - description of 31
- UIS_CREATE 28
 - description of 31
- UIS_FLAGS
 - checking with inline functions 45
- UIS_OPENCREATE 28
 - description of 31
- UIS_READ 28
 - description of 31
- UIS_READWRITE 28
 - description of 31
- UIS_TEMPORARY 28
- UIS_TEXT 28
 - description of 31
- UIW_BIGNUM
 - information requests in 62
 - NMF_SCIENMTC flag in 62
- UIW_BUTTON 141
 - bottom edge of 63
 - toggling appearance of 63
 - toggling behavior in 62
- UIW_COMBO_BOX 161
 - virtualized Add() and Subtract() in 77
- UIW_DATE
 - information requests 63
- UIW_EDIT_STRING 150
- UIW_ELLIPSE 210
- UIW_GEOMETRY_OBJECT 210
- UIW_GROUP 101,118,156
- UIW_HZ_LIST 156,161
 - virtualized Add() and Subtract() in 77
- UIW_IMAGE
 - constructing 22
 - creating persistent object in 24
 - destroying 22
 - overview of 17
- UIW_POP_UPJTEM
 - MNIF_ABOUT
 - using with Macintosh 63
- UIW_POP_UP_MENU 156
 - virtualized Add() and Subtract() in 77
- UIW_PULL_DOWN_MENU
 - virtualized Add() and Subtract() in 77
- UIW_REAL
 - flags in 63
- UIW_SCROLL_BAR
 - default width and height 64

- UIW_STATUS_BAR
 - adding objects to 64
 - virtualized Add() and Subtract() in 77
- UIW_STRING 126
 - inheritance of 55
- UIW_SYSTEM_BUTTON
 - using with Macintosh 64
- UIW_TABLE 138, 145, 146, 203, 205
 - building with persistence 64
 - DataSet() 71
 - DeleteRecord() 213
 - edit mode 203
 - features of 71
 - InsertRecord() 213
 - manipulating visually 72
 - scrolling more than 32767 records in 158
 - using with a database 219
 - using with third-party database 213
- UIW_TABLE_RECORD 203, 204, 205, 207
- UIW_TEXT 204
- UIW_TIME
 - information requests 64
- UIW_VT_LIST 156, 161
 - virtualized Add() and Subtract() in 77
- UIW_WINDOW 156, 161, 168, 212, 218
 - virtualized Add() and Subtract() 77
- Unicode 150
 - character map table 230
 - China 233
 - eliminating 227
 - Greek 232
 - Japan 233
 - Japanese 232
 - Korea 233
 - Korean 233
 - Taiwan 233
 - using with NormalizeString() 52
- Unicode tables 230
- UNICODE.DAT 230
 - decreasing size of 230
- United States 233
- Unlink()
 - definition of 29
 - description of 33
- US CP 437 230
- userfunction 205
- user function 216
 - using as member of a class 216
 - using with drag-and-drop 199
- user functions
 - using those that are members of a class 216
- userFunction 216
- userObject 168
- userTable 25
 - using with UIW_IMAGE 24

V

- validation
 - of a field 168
- VALIDT
 - description of 68
- vertical list 161
 - columns in a 160
 - deriving a new class with columns 160
 - with buttons 195
- vertical list with buttons 195
- VirtualGet() 110, 120
 - using with UIJDISPLAY 56
- VirtualPut() 110
 - using with UI.DISPLAY 56
- Visual C++ 83

W

- Watcom 77
- Watcom IDE for Windows 85, 111, 129, 134, 152, 157, 158, 170,175,223
- window 161,164
 - closing an MDI child 154
 - deriving a 212
 - displaying temporarily at startup 218
 - how to close a 154
- Window Editor 144
- WindowManager 96,103,166, 218
- window object
 - how to use 72
 - actions
 - checking with inline functions 46
- windowID 101
- windowManager 98, 122, 154, 215, 237
- windowManager- 198
- Windows 98, 164, 195
 - ANSI/Latin I 231
 - Arabic 231
 - argc and argv in UI_APPLICATION 55
 - Cyrillic 231
 - Eastern Europe 231
 - fixed-width fonts 94
 - Greek 231
 - Hebrew 231
 - Japanese 230
 - stack size with 78
 - Taiwan 230
 - Turkish 231
 - using help system with OpenZinc applications 237
 - Watcom IDE 85
- windows
 - without titles 164
- WinDrawBitmap 120, 121
- WinGetPS 120, 121
- WinHelp() 239

- WinMessageBox() 98
- WNF_AUTO_SELECT 165
- WNF_FLAGS
 - checking with inline functions 46
- WNF_SELECT_MULTIPLE 156
- WOAF_ACCEPTS_DROP 199, 201, 202
 - description of 74
- WOAF_COPY_DRAG_OBJECT 201, 202
 - description of 74
- WOAF_DRAG_OBJECT 202
- WOAF_FLAGS
 - checking with inline functions 46
- WOAF_MDI_OBJECT
 - setting 65
- WOAF_MOVE_DRAG_OBJECT 201
 - description of 74
- WOAF_NO_DESTROY 167
- WOF_FLAGS
 - checking with inline functions 46
- WOF_NO_ALLOCATE_DATA 138, 139, 145, 213
- WOF_NON_FIELD_REGION
 - cleared during construction of UIW_WINDOW 65
- WOF_PIXEL
 - with UI_WINDOW_OBJECT 61
- WOS_FLAGS
 - checking with inline functions 47
- WOS_GRAPHICS 166
- WOS_OWNERDRAW 110, 117
- WOS_READ_ERROR 140
- WOS_SELECTED 165
- Write()
 - defined with ZIL_UNICODE 30
 - definitions of 29
 - description of 35
 - using with Unicode 36

Z

- Z_ERROR
 - type definition 28
- z_map.cpp 101
- z_win2.cpp 101
- ZAF_MESSAGE_WINDOW
 - member variables in 66
 - new member variables 66
- ZAF_SERVICE_MANAGER 149
- ZAF_STRING_EDITOR 148, 149, 150
- ZIL_BITMAP_HANDLE 25
- ZIL_COLOR 211
- ZIL_DECORATION_MANAGER
 - defaultOSName 61
- ZIL_FARDATA 238
- ZIL_FILE
 - destroying a 33
- ZIL_FILE()

- creating a new file with 33
- ZIL_I18N_MANAGER
 - defaultOSName 61
- ZIL_ICHAR 146, 238, 239
- ZIL_IMAGE_HANDLE 20
- ZIL_LANGUAGE_MANAGER
 - defaultOSName 61
- ZIL_LOAD 206
 - defined in OpenZinc Designer 73
 - defining in UIW_IMAGE 23
- ZIL_LOCALE_MANAGER
 - defaultOSName 62
- ZIL_SCREENID 238
- ZIL_STORAGE_OBJECT_READ_ONLY 239
 - used in UIW_IMAGE 25
 - used with UIW_IMAGE 24
- ZIL_STORAGE_READ_ONLY
 - used in UIW_IMAGE 24, 25
 - used with UIW_IMAGE 24
- ZIL_STORE
 - defined in OpenZinc Designer 73
 - using with UIW_IMAGE 20
- ZIL_UNICODE
 - if defined in ZIL_FILE 30
- OpenZinc
 - upgrading to 1.0 from earlier versions 175
- OpenZinc 3.5
 - displaying MDI child windows in 208
- OpenZinc 3.6
 - displaying MDI child windows in 208
 - OpenZinc Designer 125, 136, 137, 141, 143, 148, 212, 237
 - displaying information in a table created with 145
 - errata in 70
 - failing to launch 143
 - Image Editor 73
 - importing and exporting files with 144
 - integrated modules in 73
 - preferences
 - using to deselect Write CPP 141
 - relative constraints in 70
 - running in graphics mode 143
 - setting drag-and-drop with 198
 - text mode 143
 - using directory services with 125
 - Window Object option in 72
- OpenZinc Designer 4.0
 - combining .CPP files created by 131
- OpenZinc events
 - mapping to system events 49
- OpenZinc libraries
 - rebuilding with PharLap 228
- OpenZinc_PRINTER
 - environment variable in UI_PRINTER 59
- ZNC 143

GNU Free Documentation License
Version 1.3, 3 November 2008

Copyright (C) 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

[<http://fsf.org/>](http://fsf.org/)

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical

connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a

section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers

- or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
 - N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
 - O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections

Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual

title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A

public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.