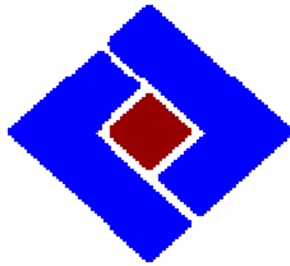


OpenZinc

Programmers Reference
Volume 2

Window Objects

Version 1

Programmer's Reference

Volume Two
Window Objects

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

INTRODUCTION 1
UI_SAMPLE_CLASS::SampleFunction
CLASSES AND STRUCTURES
INCLUDE FILE HIERARCHY
CLASS HIERARCHY

CHAPTER 1 - UIW_BIGNUM 15
General Members
UIW_BIGNUM::UIW_BIGNUM
UIW_BIGNUM::~UIW_BIGNUM
UIW_BIGNUM::ClassName
UIW_BIGNUM::DataGet
UIW_BIGNUM::DataSet
UIW_BIGNUM::Event
UIW_BIGNUM::Information
UIW_BIGNUM::SetLanguage
UIW_BIGNUM::Validate
Storage Members
UIW_BIGNUM::UIW_BIGNUM
UIW_BIGNUM::Load
UIW_BIGNUM::New
UIW_BIGNUM::NewFunction
UIW_BIGNUM::Store

CHAPTER 2 - UIW_BORDER 39
General Members
UIW_BORDER::UIW_BORDER
UIW_BORDER::ClassName
UIW_BORDER::DataGet
UIW_BORDER::DataSet
UIW_BORDER::DrawItem
UIW_BORDER::Event
UIW_BORDER::Information
Storage Members
UIW_BORDER::UIW_BORDER
UIW_BORDER::Load
UIW_BORDER::New
UIW_BORDER::NewFunction

UIW_BORDER::Store

CHAPTER 3 - UIW_BUTTON **55**
General Members
UIW_BUTTON::UIW_BUTTON
UIW_BUTTON::~UIW_BUTTON
UIW_BUTTON::ClassName
UIW_BUTTON::DataGet
UIW_BUTTON::DataSet
UIW_BUTTON::DrawItem
UIW_BUTTON::Event
UIW_BUTTON::Information
UIW_BUTTON::Message
UIW_BUTTON::SetDecorations
Storage Members
UIW_BUTTON::UIW_BUTTON
UIW_BUTTON::Load
UIW_BUTTON::New
UIW_BUTTON::NewFunction
UIW_BUTTON::Store

CHAPTER 4 - UIW_COMBO_BOX **85**
General Members
UIW_COMBO_BOX::UIW_COMBO_BOX
UIW_COMBO_BOX::~UIW_COMBO_BOX
UIW_COMBO_BOX::Add
UIW_COMBO_BOX::operator +
UIW_COMBO_BOX::ClassName
UIW_COMBO_BOX::Count
UIW_COMBO_BOX::Current
UIW_COMBO_BOX::Destroy
UIW_COMBO_BOX::Event
UIW_COMBO_BOX::First
UIW_COMBO_BOX::Get
UIW_COMBO_BOX::Index
UIW_COMBO_BOX::Information
UIW_COMBO_BOX::Last
UIW_COMBO_BOX::Sort
UIW_COMBO_BOX::Subtract
UIW_COMBO_BOX::operator -
Storage Members
UIW_COMBO_BOX::UIW_COMBO_BOX
UIW_COMBO_BOX::Load

UIW_COMBO_BOX::New
UIW_COMBO_BOX::NewFunction
UIW_COMBO_BOX::Store

CHAPTER 5 - UIW_DATE. 113
General Members
UIW_DATE::UIW_DATE
UIW_DATE::~UIW_DATE
UIW_DATE::ClassName
UIW_DATE::DataGet
UIW_DATE::DataSet
UIW_DATE::Event
UIW_DATE::Information
UIW_DATE::SetLanguage
UIW_DATE::Validate
Storage Members
UIW_DATE::UIW_DATE
UIW_DATE::Load
UIW_DATE::New
UIW_DATE::Store

CHAPTER 6 - UIW_FORMATTED_STRING. 137
General Members
UIW_FORMATTED_STRING::UIW_FORMATTED_STRING
UIW_FORMATTED_STRING::~UIW_FORMATTED_STRING
UIW_FORMATTED_STRING::ClassName
UIW_FORMATTED_STRING::DataGet
UIW_FORMATTED_STRING::DataSet
UIW_FORMATTED_STRING::Event
UIW_FORMATTED_STRING::Export
UIW_FORMATTED_STRING::Import
UIW_FORMATTED_STRING::Information
Storage Members
UIW_FORMATTED_STRING::UIW_FORMATTED_STRING
UIW_FORMATTED_STRING::Load
UIW_FORMATTED_STRING::New
UIW_FORMATTED_STRING::NewFunction
UIW_FORMATTED_STRING::Store

CHAPTER 7 - UIW_GROUP. 161
General Members
UIW_GROUP::UIW_GROUP
UIW_GROUP::~UIW_GROUP

UIW_GROUP::ClassName
UIW_GROUP::DataGet
UIW_GROUP::DataSet
UIW_GROUP::Event
UIW_GROUP::Information
UIW_GROUP::RegionMax
Storage Members
UIW_GROUP::UIW_GROUP
UIW_GROUP::Load
UIW_GROUP::New
UIW_GROUP::NewFunction
UIW_GROUP::Store

CHAPTER 8 - UIW_HZ_LIST. 181
General Members
UIW_HZ_LIST::UIW_HZ_LIST
UIW_HZ_LIST::~UIW_HZ_LIST
UIW_HZ_LIST::Add
UIW_HZ_LIST::ClassName
UIW_HZ_LIST::Destroy
UIW_HZ_LIST::Event
UIW_HZ_LIST::Information
UIW_HZ_LIST::ScrollEvent
UIW_HZ_LIST::Sort
UIW_HZ_LIST::Subtract
UIW_HZ_LIST::operator -
Storage Members
UIW_HZ_LIST::UIW_HZ_LIST
UIW_HZ_LIST::Load
UIW_HZ_LIST::New
UIW_HZ_LIST::NewFunction
UIW_HZ_LIST::Store

CHAPTER 9 - UIW_ICON. 207
General Members
UIW_ICON::UIW_ICON
UIW_ICON::UIW_ICON
UIW_ICON::ClassName
UIW_ICON::DataGet
UIW_ICON: .DataSet
UIW_ICON::Drawltem
UIW_ICON::Event
UIW_ICON::Information

Storage Members
UIW_ICON::UIW_ICON
UIW_ICON::Load
UIW_ICON::New
UIW_ICON::NewFunction
UIW_ICON::Store

CHAPTER 10 - UIW_INTEGER 231
General Members
UIW_INTEGER::UIW_INTEGER
UIW_INTEGER::~UIW_INTEGER
UIW_INTEGER::ClassName
UIW_INTEGER::DataGet
UIW_INTEGER::DataSet
UIW_INTEGER::Event
UIW_INTEGER::Information
UIW_INTEGER::SetLanguage
UIW_INTEGER::Validate
Storage Members
UIW_INTEGER::UIW_INTEGER
UIW_INTEGER::Load
UIW_INTEGER::New
UIW_INTEGER::NewFunction
UIW_INTEGER::Store

CHAPTER 11 - UIW_MAXIMIZE_BUTTON 251
General Members
UIW_MAXIMIZE_BUTTON::UIW_MAXIMIZE_BUTTON
UIW_MAXIMIZE_BUTTON::~UIW_MAXIMIZE_BUTTON
UIW_MAXIMIZE_BUTTON::ClassName
UIW_MAXIMIZE_BUTTON::Event
UIW_MAXIMIZE_BUTTON::Information
UIW_MAXIMIZE_BUTTON::SetDecorations
Storage Members
UIW_MAXIMIZE_BUTTON::UIW_MAXIMIZE_BUTTON
UIW_MAXIMIZE_BUTTON::Load
UIW_MAXIMIZE_BUTTON::New
UIW_MAXIMIZE_BUTTON::NewFunction
UIW_MAXIMIZE_BUTTON::Store

CHAPTER 12 - UIW_MINIMIZE_BUTTON 265
General Members
UIW_MINIMIZE_BUTTON::UIW_MINIMIZE_BUTTON

UIW_MINIMIZE_BUTTON::~UIW_MINIMIZE_BUTTON
UIW_MINIMIZE_BUTTON::ClassName
UIW_MINIMIZE_BUTTON::Event
UIW_MINIMIZE_BUTTON::Information
UIW_MINIMIZE_BUTTON::SetDecorations
Storage Members
UIW_MINIMIZE_BUTTON::UIW_MINIMIZE_BUTTON
UIW_MINIMIZE_BUTTON::Load
UIW_MINIMIZE_BUTTON::New
UIW_MINIMIZE_BUTTON::NewFunction
UIW_MINIMIZE_BUTTON::Store

CHAPTER 13 - UIW_NOTEBOOK. 279
General Members
UIW_NOTEBOOK::UIW_NOTEBOOK
UIW_NOTEBOOK::~UIW_NOTEBOOK
UIW_NOTEBOOK::Add
UIW_NOTEBOOK::operator +
UIW_NOTEBOOK::ClassName
UIW_NOTEBOOK::DrawItem
UIW_NOTEBOOK::Event
UIW_NOTEBOOK::Information
Storage Members
UIW_NOTEBOOK::UIW_NOTEBOOK
UIW_NOTEBOOK::Load
UIW_NOTEBOOK::New
UIW_NOTEBOOK::NewFunction
UIW_NOTEBOOK::Store

CHAPTER 14 - UIW_POP_UP_ITEM. 295
General Members
UIW_POP_UP_ITEM::UIW_POP_UP_ITEM
UIW_POP_UP_ITEM::~UIW_POP_UP_ITEM
UIW_POP_UP_ITEM::Add
UIW_POP_UP_ITEM::operator +
UIW_POP_UP_ITEM::ClassName
UIW_POP_UP_ITEM::DrawItem
UIW_POP_UP_ITEM::Event
UIW_POP_UP_ITEM::Information
UIW_POP_UP_ITEM::SetDecorations
UIW_POP_UP_ITEM: Subtract
UIW_POP_UP_ITEM::operator -
Storage Members

UIW_POP_UP_ITEM::UIW_POP_UP_ITEM
UIW_POP_UP_ITEM::Load
UIW_POP_UP_ITEM::New
UIW_POP_UP_ITEM::NewFunction
UIW_POP_UP_ITEM::Store

CHAPTER 15 - UIW_POP_UP_MENU 319
General Members
UIW_POP_UP_MENU::UIW_POP_UP_MENU
UIW_POP_UP_MENU::~UIW_POP_UP_MENU
UIW_POP_UP_MENU::Add
UIW_POP_UP_MENU::ClassName
UIW_POP_UP_MENU::Event
UIW_POP_UP_MENU::Information
UIW_POP_UP_MENU::Subtract
UIW_POP_UP_MENU::operator -
Storage Members
UIW_POP_UP_MENU::UIW_POP_UP_MENU
UIW_POP_UP_MENU::Load
UIW_POP_UP_MENU::New
UIW_POP_UP_MENU::NewFunction
UIW_POP_UP_MENU::Store

CHAPTER 16 - UIW_PROMPT 337
General Members
UIW_PROMPT::UIW_PROMPT
UIW_PROMPT: :UIW_PROMPT
UIW_PROMPT::ClassName
UIW_PROMPT::DataGet
UIW_PROMPT::DataSet
UIW_PROMPT::Drawltem
UIW_PROMPT::Event
UIW_PROMPT::Information
Storage Members
UIW_PROMPT::UIW_PROMPT
UIW_PROMPT::Load
UIW_PROMPT::New
UIW_PROMPT::NewFunction
UIW_PROMPT::Store

CHAPTER 17 - UIW_PULL_DOWN_ITEM 357
General Members
UIW_PULL_DOWN_ITEM::UIW_PULL_DOWN_ITEM

UIW_PULL_DOWN_ITEM::~UIW_PULL_DOWN_ITEM
UIW_PULL_DOWN_ITEM::Add
UIW_PULL_DOWN_ITEM::operator +
UIW_PULL_DOWN_ITEM::ClassName
UIW_PULL_DOWN_ITEM::DrawItem
UIW_PULL_DOWN_ITEM::Event
UIW_PULL_DOWN_ITEM::Information
UIW_PULL_DOWN_ITEM::Subtract
UIW_PULL_DOWN_ITEM::operator -
Storage Members
UIW_PULL_DOWN_ITEM::UIW_PULL_DOWN_ITEM
UIW_PULL_DOWN_ITEM::Load
UIW_PULL_DOWN_ITEM::New
UIW_PULL_DOWN_ITEM::NewFunction
UIW_PULL_DOWN_ITEM::Store

CHAPTER 18 - UIW_PULL_DOWN_MENU. 377
General Members
UIW_PULL_DOWN_MENU::UIW_PULL_DOWN_MENU
UIW_PULL_DOWN_MENU::~UIW_PULL_DOWN_MENU
UIW_PULL_DOWN_MENU::Add
UIW_PULL_DOWN_MENU::ClassName
UIW_PULL_DOWN_MENU::Event
UIW_PULL_DOWN_MENU::Information
UIW_PULL_DOWN_MENU::ItemDepthSearch
UIW_PULL_DOWN_MENU::Subtract
UIW_PULL_DOWN_MENU::operator -
Storage Members
UIW_PULL_DOWN_MENU::UIW_PULL_DOWN_MENU
UIW_PULL_DOWN_MENU::Load
UIW_PULL_DOWN_MENU::New
UIW_PULL_DOWN_MENU::NewFunction
UIW_PULL_DOWN_MENU::Store

CHAPTER 19 - UIW_REAL. 395
General Members
UIW_REAL::UIW_REAL
UIW_REAL::~UIW_REAL
UIW_REAL::ClassName
UIW_REAL::DataGet
UIW_REAL::DataSet
UIW_REAL::Event
UIW_REAL::Format

UIW_REAL::Information
UIW_REAL::SetLanguage
UIW_REAL::Validate
Storage Members
UIW_REAL::UIW_REAL
UIW_REAL::Load
UIW_REAL::New
UIW_REAL::NewFunction
UIW_REAL::Store

CHAPTER 20 - UIW_SCROLL_BAR. 417
General Members
UIW_SCROLL_BAR::UIW_SCROLL_BAR
UIW_SCROLL_BAR::~UIW_SCROLL_BAR
UIW_SCROLL_BAR::ClassName
UIW_SCROLL_BAR::DrawItem
UIW_SCROLL_BAR::Event
UIW_SCROLL_BAR::Information
Storage Members
UIW_SCROLL_BAR::UIW_SCROLL_BAR
UIW_SCROLL_BAR::Load
UIW_SCROLL_BAR::New
UIW_SCROLL_BAR::NewFunction
UIW_SCROLL_BAR::Store

CHAPTER 21 - UIW_SPIN_CONTROL. 441
General Members
UIW_SPIN_CONTROL::UIW_SPIN_CONTROL
UIW_SPIN_CONTROL::UIW_SPIN_CONTROL
UIW_SPIN_CONTROL::ClassName
UIW_SPIN_CONTROL::Event
UIW_SPIN_CONTROL::Information
Storage Members
UIW_SPIN_CONTROL::UIW_SPIN_CONTROL
UIW_SPIN_CONTROL::Load
UIW_SPIN_CONTROL::New
UIW_SPIN_CONTROL::NewFunction
UIW_SPIN_CONTROL::Store

CHAPTER 22 - UIW_STATUS_BAR. 459
General Members
UIW_STATUS_BAR::UIW_STATUS_BAR
UIW_STATUS_BAR::~UIW_STATUS_BAR

UIW_STATUS_BAR::ClassName
UIW_STATUS_BAR::DrawItem
UIW_STATUS_BAR::Event
UIW_STATUS_BAR::Information
Storage Members
UIW_STATUS_BAR::UIW_STATUS_BAR
UIW_STATUS_BAR::Load
UIW_STATUS_BAR::New
UIW_STATUS_BAR::NewFunction
UIW_STATUS_BAR::Store

CHAPTER 23 - UIW_STRING 473
General Members
UIW_STRING::UIW_STRING
UIW_STRING::~UIW_STRING
UIW_STRING::ClassName
UIW_STRING::DataGet
UIW_STRING::DataSet
UIW_STRING::DrawItem
UIW_STRING::Event
UIW_STRING::Information
UIW_STRING::ParseRange
Storage Members
UIW_STRING::UIW_STRING
UIW_STRING::Load
UIW_STRING::New
UIW_STRING::NewFunction
UIW_STRING::Store

CHAPTER 24 - UIW_SYSTEM_BUTTON 499
General Members
UIW_SYSTEM_BUTTON::UIW_SYSTEM_BUTTON
UIW_SYSTEM_BUTTON::~UIW_SYSTEM_BUTTON
UIW_SYSTEM_BUTTON::ClassName
UIW_SYSTEM_BUTTON::Event
UIW_SYSTEM_BUTTON::Generic
UIW_SYSTEM_BUTTON::Information
UIW_SYSTEM_BUTTON::SetDecorations
UIW_SYSTEM_BUTTON::SetLanguage
Storage Members
UIW_SYSTEM_BUTTON::UIW_SYSTEM_BUTTON
UIW_SYSTEM_BUTTON::Load
UIW_SYSTEM_BUTTON::New

UIW_SYSTEM_BUTTON::NewFunction
UIW_SYSTEM_BUTTON::Store

CHAPTER 25 - UIW_TABLE **519**
General Members
UIW_TABLE::UIW_TABLE
UIW_TABLE::~UIW_TABLE
UIW_TABLE::DataGet
UIW_TABLE::DataSet
UIW_TABLE::DeleteRecord
UIW_TABLE::DrawItem
UIW_TABLE::DrawRecord
UIW_TABLE::Event
UIW_TABLE::GetRecord
UIW_TABLE::Information
UIW_TABLE::InsertRecord
UIW_TABLE::SetCurrent
Storage Members
UIW_TABLE::UIW_TABLE
UIW_TABLE::Load
UIW_TABLE::New
UIW_TABLE::NewFunction
UIW_TABLE::Store

CHAPTER 26 - UIW_TABLE_HEADER **541**
General Members
UIW_TABLE_HEADER::UIW_TABLE_HEADER
UIW_TABLE_HEADER::~UIW_TABLE_HEADER
UIW_TABLE_HEADER::DrawItem
UIW_TABLE_HEADER::Event
UIW_TABLE_HEADER::Information
Storage Members
UIW_TABLE_HEADER::UIW_TABLE_HEADER
UIW_TABLE_HEADER::Load
UIW_TABLE_HEADER::New
UIW_TABLE_HEADER::NewFunction
UIW_TABLE_HEADER::Store

CHAPTER 27 - UIW_TABLE_RECORD **555**
General Members
UIW_TABLE_RECORD::UIW_TABLE_RECORD
UIW_TABLE_RECORD::DrawItem
UIW_TABLE_RECORD::Event

UIW_TABLE_RECORD::Information
UIW_TABLE_RECORD::RegionMax
UIW_TABLE_RECORD::VirtualRecord
Storage Members
UIW_TABLE_RECORD::UIW_TABLE_RECORD
UIW_TABLE_RECORD::Load
UIW_TABLE_RECORD::New
UIW_TABLE_RECORD::NewFunction
UIW_TABLE_RECORD::Store

CHAPTER 28 - UIW_TEXT. 571
General Members
UIW_TEXT::UIW_TEXT
UIW_TEXT::~UIW_TEXT
UIW_TEXT::ClassName
UIW_TEXT::CursorOffset
UIW_TEXT::DataGet
UIW_TEXT::DataSet
UIW_TEXT::DrawItem
UIW_TEXT::Event
UIW_TEXT::GetCursorPos
UIW_TEXT::Information
UIW_TEXT::SetCursorPos
Storage Members
UIW_TEXT::UIW_TEXT
UIW_TEXT::Load
UIW_TEXT::New
UIW_TEXT::NewFunction
UIW_TEXT::Store

CHAPTER 29 - UIW_TIME. 597
General Members
UIW_TIME::UIW_TIME
UIW_TIME::~UIW_TIME
UIW_TIME::ClassName
UIW_TIME::DataGet
UIW_TIME::DataSet
UIW_TIME::Event
UIW_TIME::Information
UIW_TIME::SetLanguage
UIW_TIME::Validate
Storage Members
UIW_TIME::UIW_TIME

UIW_TIME::Load
UIW_TIME::New
UIW_TIME::NewFunction
UIW_TIME::Store

CHAPTER 30 - UIW_TITLE 621
General Members
UIW_TITLE::UIW_TITLE
UIW_TITLE::~UIW_TITLE
UIW_TITLE::ClassName
UIW_TITLE::DataGet
UIW_TITLE::DataSet
UIW_TITLE::Event
UIW_TITLE::Information
Storage Members
UIW_TITLE::UIW_TITLE
UIW_TITLE::Load
UIW_TITLE::New
UIW_TITLE::NewFunction
UIW_TITLE::Store

CHAPTER 31 - UIW_TOOL_BAR 637
General Members
UIW_TOOL_BAR::UIW_TOOL_BAR
UIW_TOOL_BAR:: UIW_TOOL_BAR
UIW_TOOL_BAR::ClassName
UIW_TOOL_BAR::Event
UIW_TOOL_BAR::Information
Storage Members
UIW_TOOL_BAR::UIW_TOOL_BAR
UIW_TOOL_BAR::Load
UIW_TOOL_BAR::New
UIW_TOOL_BAR::NewFunction
UIW_TOOL_BAR::Store

CHAPTER 32 - UIW_VT_LIST 653
General Members
UIW_VT_LIST::UIW_VT_LIST
UIW_VT_LIST::~UIW_VT_LIST
UIW_VT_LIST::Add
UIW_VT_LIST::ClassName
UIW_VT_LIST::Destroy
UIW_VT_LIST::Event

UIW_VT_LIST::Information
UIW_VT_LIST::RegionMax
UIW_VT_LIST::ScrollEvent
UIW_VT_LIST::Sort
UIW_VT_LIST: Subtract
UIW_VT_LIST::operator -
UIW_VT_LIST::TopWidget
Storage Members
UIW_VT_LIST::UIW_VT_LIST
UIW_VT_LIST::Load
UIW_VT_LIST::New
UIW_VT_LIST::NewFunction
UIW_VT_LIST::Store

CHAPTER 33 - UIW_WINDOW..................................679
General Members
UIW_WINDOW::UIW_WINDOW
UIW_WINDOW::~UIW_ WINDOW
UIW_WINDOW::Add
UIW_WINDOW::operator +
UIW_WINDOW::CheckSelection
UIW_WINDOW::ClassName
UIW_WINDOW::Current
UIW_WINDOW::Destroy
UIW_WINDOW::DrawItem
UIW_WINDOW::Event
UIW_WINDOW::First
UIW_WINDOW::Generic
UIW_WINDOW::Get
UIW_WINDOW::Information
UIW_WINDOW::Last
UIW_WINDOW::RegionMax
UIW_WINDOW::ScrollEvent
UIW_WINDOW::SetLanguage
UIW_WINDOW::StringCompare
UIW_WINDOW::Subtract
UIW_WINDOW::operator -
Storage Members
UIW_WINDOW::UIW_WINDOW
UIW_WINDOW::Load
UIW_WINDOW::New
UIW_WINDOW::NewFunction
UIW_WINDOW::Store

CHAPTER 34 - ZAF_DIALOG_WINDOW..........................721
General Members
ZAF_DIALOG_WINDOW::ZAF_DIALOG_WINDOW
ZAF_DIALOG_WINDOW::~ZAF_DIALOG_WINDOW
ZAF_DIALOG_WINDOW::Control
Storage Members
ZAF_DIALOG_WINDOW::ZAF_DIALOG_WINDOW
ZAF_DIALOG_WINDOW::Load
ZAF_DIALOG_WINDOW::New
ZAF_DIALOG_WINDOW::Store

CHAPTER 35 - **ZAF_MESSAGE_WINDOW. 733**
General Members
ZAF_MESSAGE_WINDOW::ZAF_MESSAGE_WINDOW
ZAF_MESSAGE_WINDOW::Control

APPENDIX A - SUPPORT DEFINITIONS. 737
Typedefs and Preprocessor Variables
FALSE
TRUE
ZIL_BAK
ZIL_BIGENDIAN
ZIL_BITMAP_HANDLE
ZIL_COLOR
ZIL_COMPARE_FUNCTION
ZIL_DECOMPOSE
ZIL_DO_FILE_118N
ZIL_DO_OS_118N
ZIL_EXIT_FUNCTION
ZIL_EXT
ZIL_HARDWARE
ZIL_HOTMARK
ZIL_IBIGNUM
ZIL_ICHAR
ZIL_ICON_HANDLE
ZIL_INT8
ZIL_INT16
ZIL_INT32
ZIL_LITTLEENDIAN
ZIL_LOAD
ZIL_MACINTOSH
ZIL_MAXPATHLEN
ZIL_MOTIF

ZIL_MOTIF_STYLE
ZIL_MSDOS
ZIL_MSWINDOWS
ZIL_MSWINDOWS_STYLE
ZIL_NEW_FUNCTION
ZIL_NUMBERID
ZIL_OLD_DEFS
ZIL_OS2
ZIL_OS2_STYLE
ZIL_PATHSEP
ZIL_POSIX
ZIL_RBIGNUM
ZIL_SCREENID
ZIL_SHADOW_BORDER
ZIL_STANDARD_BORDER
ZIL_STORE
ZIL_TEXT_ONLY
ZIL_3D_BORDER
ZIL_3x_COMPAT
ZIL_UINT8
ZIL_UINT16
ZIL_UINT32
ZIL_UNICODE
ZIL_USER_FUNCTION
ZIL_WINNT
Macros
AbsValue
attrib
FlagSet
FlagsSet
HIWORD
LOWORD
MaxValue
MinValue
ZIL_NULLF
ZIL_NULLH
ZIL_NULLP
ZIL_VOIDF
ZIL_VOIDP

APPENDIX B - SYSTEM EVENTS **757**

APPENDIX C - LOGICAL EVENTS **763**

APPENDIX D - CLASS IDENTIFIERS **769**

APPENDIX E - OpenZinc OBJECT STORAGE **773**

APPENDIX F - CHARACTER SETS **785**

APPENDIX G - ISO COUNTRY CODES **789**
Country/Locale Codes

APPENDIX H - ISO LANGUAGE CODES **797**
Language Codes

APPENDIX | - HARDWARE ISSUES **803**
Binding Device Drivers
Macros
Generic Keyboard Functions
I_KeyboardClose
I_KeyboardOpen
I_KeyboardQuery
I_KeyboardRead
Generic Mouse Functions
I_MouseClose
I_MouseOpen
Global Variables
Text Driver Functions
I_ScreenClose
I_ScreenOpen
I_ScreenPut
I_CursorPosition
I_CursorRemove
Generic Internationalization Functions
I_GetCodePage

INDEX **813**

INTRODUCTION

The *Programmer's Reference Volume* 2 contains descriptions of OpenZinc Application Framework classes, the calling conventions used to invoke the class member functions, short code samples using the class member functions, and information about other related classes or example programs.

Some miscellaneous information is presented in the Appendices. This section (Appendices A through |) contains support definitions, system event definitions, logical event definitions, class identifications, storage information, internationalization information, and some hardware issues.

UI_SAMPLE_CLASS::SampleFunction

Syntax

returnValue SampleFunction(typel *parameter 1*, type2 **parameter2);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

NOTE: A blackened box indicates a supported environment.

Remarks

A brief description of what **SampleFunction**() does.

- $returnValue_{out}$ gives a complete description of the return value. The subscript "out" indicates that the variable (the return value in this case) does not require an initial value and that it receives a value from the function.

- $parameter1_{in}$ gives a complete description of function parameter 1. The subscript "in" indicates that the variable requires an initial value and that it is not changed by the function.

- $parameter2_{in/out}$ gives a complete description of function parameter 2. The subscript "in/out" indicates that the variable requires an initial value, but that it may also receive a different value from the function.

Example

This section provides a coding example of how **SampleFunction**() was used in the development of other library functions or development utilities. The function itself often appears in bold type within the example code.

CLASSES AND STRUCTURES

General purpose

```
attrib
FlagSet
FlagsSet
MaxValue
MinValue
ZIL_NULLF
ZIL_NULLH
ZIL_NULLP
TRUE
FALSE
ZIL_INT8
ZIL_UINT8
ZIL_INT16
ZIL_UINT16
ZIL_INT32
ZIL_UINT32
ZIL_VOIDF
ZIL_VOIDP

struct UI_ITEM

class UI_APPLICATION
class UI_ELEMENT
class UI_LIST
class UI_LIST_BLOCK
class UI_PATH
class UI_PATH_ELEMENT
class ZIL_BIGNUM
class ZIL_DATE
class ZIL_TIME
class ZIL_UTIME
```

Error system

```
    class UI_ERROR_STUB
class     UI_ERROR_SYSTEM
```

Event management

```
struct UI_EVENT
struct UI_EVENT_MAP
struct UI_KEY
struct UI_POSITION
struct UI_REGION
struct UI_SCROLL_INFORMATION

class UI_DEVICE
class UI_EVENT_MANAGER
class UI_QUEUE_BLOCK
class UI_QUEUE_ELEMENT
class UID_CURSOR
class U1D_KEYB0ARD
class UID_MOUSE
class UID_TIMER
```

Help system

```
class UI_HELP_STUB
class UI_HELP_SYSTEM
```

Internationalization

```
struct ZIL_BITMAP_ELEMENT
struct ZIL_LOCALE_ELEMENT
struct ZIL_LANGUAGE_ELEMENT
struct ZIL_TEXT_ELEMENT

class ZIL_DECORATION
class ZIL_DECORATION_MANAGER
class ZIL_I18N
class ZIL_I18N_MANAGER
class ZIL_INTERNATIONAL
class ZIL_LANGUAGE
class ZIL_LANGUAGE_MANAGER
class ZIL_LOCALE
class ZIL_LOCALE_MANAGER
class ZIL_MAP_CHARS
```

Printer

```
class UI_PRINTER
```

Screen display

```
struct UI_PALETTE
struct UI_PALETTE_MAP
struct UI_POSITION
struct UI_REGION

class UI_BGI_DISPLAY
class UI_DISPLAY
class UI_GRAPHICS_DISPLAY
class UI_MACINTOSH_DISPLAY
class UI_MSC_DISPLAY
class UI_MSWINDOWS_DISPLAY
class UI_NEXTSTEP_DISPLAY
class UI_OS2_DISPLAY
class UI_REGION_ELEMENT
class UI_REGION_LIST
class UI_TEXT_DISPLAY
class UI_WCC_DISPLAY
class UI_XT_DISPLAY
```

Storage

```
class ZIL_DELTA_STORAGE_OBJECT
class ZIL_DELTA_STORAGE_OBJECT_READ_ONLY
class ZIL_STORAGE
class ZIL_STORAGE_DIRECTORY
class ZIL_STORAGE_OBJECT
```

```
class  ZIL_STORAGE_OBJECT_READ_ONLY
class  ZIL_STORAGE_READ_ONLY
```

Window management

```
struct  UI_SCROLL_INFORMATION

class UI_ATTACHMENT
class UI_CONSTRAINT
class UI_DIMENSION_CONSTRAINT
class UI_GEOMETRY_MANAGER
class UI_RELATIVE_CONSTRAINT
class UI_WINDOW_MANAGER
class UI_WINDOW_OBJECT
class UIW_BIGNUM
class UIW_BORDER
class UIW_BUTTON
class UIW_COMBO_BOX
class UIW_DATE
class UIW_FORMATTED_STRING
class UIW_GROUP
class UIW_HZ_LIST
class UIW_ICON
class UIW_INTEGER
class UIW_MAXIMIZE_BUTTON
class UIW_MINIMIZE_BUTTON
class UIW_NOTEBOOK
class UIW_POP_UP_ITEM
class UIW_POP_UP_MENU
class UIW_PROMPT
class UIW_PULL_DOWN_ITEM
class UIW_PULL_DOWN_MENU
class UIW_REAL
class UIW_SCROLL_BAR
class UIW_SPIN_CONTROL
class UIW_STATUS_BAR
class UIW_STRING
class UIW_SYSTEM_BUTTON
class UIW_TABLE
class UIW_TABLE_HEADER
class UIW_TABLE_RECORD
class UIW_TEXT
class UIW_TIME
class UIW_TITLE
class UIW_TOOL_BAR
class UIW_VT_LIST
class UIW_WINDOW
class ZAF_DIALOG_WINDOW
class ZAF_MESSAGE_WINDOW
```

INCLUDE FILE HIERARCHY

ULENV.HPP

```
// Version information
// General OpenZinc Switches
// Optimization switches for various compiler problems.
// Presentation switches for the library.
// Switches for the international language versions.
// Compiler/Environment Default Dependencies
// ZIL_NULLP, ZIL_NULLF, ZIL_NULLH, ZIL_V01DF, ZIL_VOIDP
// BORLAND
// MICROSOFT
// IBM
// SYMANTEC & ZORTECH
// WATCOM
// DJGPP, GNU C++ port DOS (1.08)
// HP-UX, CC (cfront from HP) and Motif
// MS-DOS, Quarterdeck DESQview/X with Motif, DJGPP G++
// SCO UNIX 3.2 with Motif or Curses
// Solaris 2.1, CC (cfront from SunPro) and Motif
// Siemens/Nixdorf SINIX and Motif
// DEC 4000 OSF/1 1.3
// Compiler/Hardware Typedef Sizes
// TRUE/FALSE
// UIF_FLAGS
// UIS_STATUS
// Macros
// Version 3.6, 3.b, 3.0 compatibility
```

UI_GEN.HPP

```
#if !defined(UI_GEN_HPP)
#   define UI_GEN_HPP
#   if !defined(UI_ENV_HPP)
#       include <ui_env.hpp>
#   endif

// ZIL_OBJECTID
// EVENT_TYPE
// ZIL_INFO_REQUEST
// UI_ELEMENT
// UI_LIST
// UI_LIST_BLOCK
// ZIL_BIT_VECTOR
// ZIL_MESSAGE
// ZIL_I18N, ZIL_LOCALE, ZIL_LANGUAGE, & ZIL_DECORATION
// ZIL_MAP_CHARS
// ZIL_INTERNATIONAL
// ZIL_BIGNUM
// NMF_FLAGS
// NMI_RESULT
// ZIL_UTIME
// ZIL_DATE
// DTF_FLAGS
// DTI_RESULT
// ZIL_TIME
// TMF_FLAGS
// TMI_RESULT
// UI_PATH_ELEMENT & UI_PATH
// ZIL_STORAGE_OBJECT & ZIL_STORAGE
// UIS_FLAGS
// ZIL_DELTA_STORAGE_OBJECT
```

```
// Version 3.6, 3.5, 3.0 compatibility
```

UI_DSP.HPP

```
#if !defined(UI_DSP_HPP)
#   define UI_DSP_HPP
#   if !defined(UI_GEN_HPP)
#       include <ui_gen.hpp>
#   endif

// ZIL_SCREENID, ZIL_BITMAP_HANDLE, ZIL_ICON_HANDLE, ZIL_SCREEN_CELL
// UI_POSITION
// UI_REGION, UI_REGION_ELEMENT, UI_REGION_LIST
// Color information
// Font information
// Image information
// Pattern information
// UI_PALETTE
// UI_DISPLAY
// UI_BGI_DISPLAY
// UI_GRAPHICS_DISPLAY
// UI_XT_DISPLAY
// UI_MSC_DISPLAY
// UI_MSWINDOWS_DISPLAY
// UI_OS2_DISPLAY
// UI_TEXT_DISPLAY
// TDM_MODE
// UI_WCC_DISPLAY
// UI_MACINTOSH_DISPLAY
// UI_NEXTSTEP_DISPLAY
// UI_PRINTER
// Version 3.6 compatibility
```

ULMAP.HPP

```
#if !defined(UI_MAP_HPP)
#   define UI_MAP_HPP
#   if !defined(UI_DSP_HPP)
#       include <ui_dsp.hpp>
#   endif

// Compiler/Environment Dependencies
// Special hotkey values
// ZIL_MSDOS
// ZIL_MSWINDOWS
// ZIL_OS2
// ZIL_X11
// ZIL_CURSES
// ZIL_MACINTOSH
// ZIL_NEXTSTEP
// Version 3.6 compatibility
```

ULEVT.HPP

```
#if !defined(UI_EVT_HPP)
#   define UI_EVT_HPP
#   if !defined(UI_DSP_HPP)
#       include <ui_dsp.hpp>
#   endif
```

```
// EVENT_TYPE
// UI_KEY
// shiftState
// Mouse Information
// UI_SCROLL_INFORMATION
// UI_EVENT
// System wide messages
// ZIL_SYSTEM_EVENT
// ZIL_LOGICAL_EVENT
// UI_DEVICE
// Device type messages
// Device state messages
// Device image messages
// UID_CURSOR
// Cursor image messages
// UID_KEYBOARD
// UID_MOUSE
// Mouse image messages
// UID_TIMER
// TMR_FLAGS
// UI_QUEUE_ELEMENT & UI_QUEUE_BLOCK
// UI_EVENT_MANAGER
// Q_FLAGS
// Version 3.6 compatibility
```

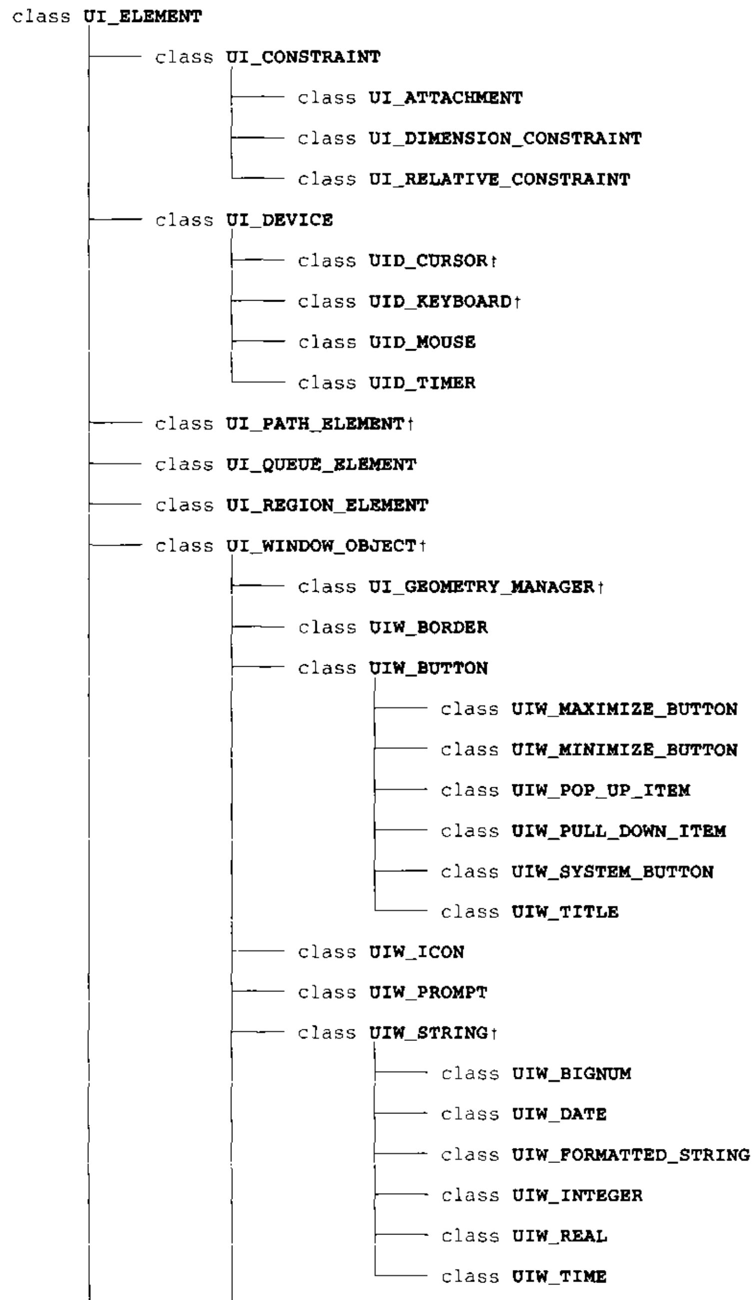
UI_WIN.HPP

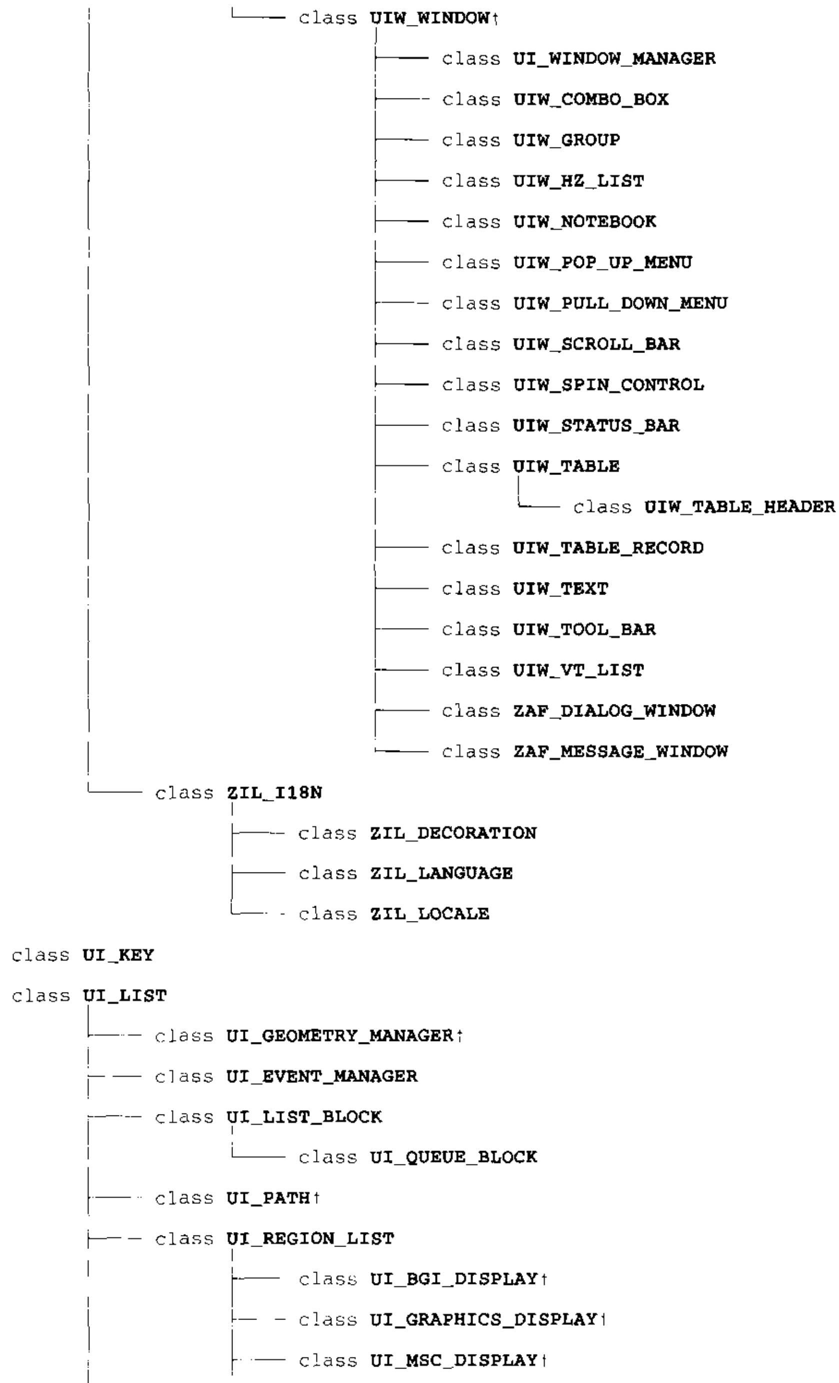
```
#if !defined(UI_WIN_HPP)
#   define UI_WIN_HPP
#   if !defined(UI_EVT_HPP)
#       include <ui_evt.hpp>
#   endif

// NUMBERID
// UI_ITEM
// Window object identifications
// ZIL_SIMPLE_OBJECTID
// ZIL_COMPLEX_OBJECTID
// ZIL_COMPOSITE_OBJECTID
// Window object system messages
// ZIL_SYSTEM_EVENT
// ZIL_LOGICAL_EVENT
// ZIL_DESIGNER_EVENT
// UI_PALETTE_MAP
// ZIL_LOGICAL_PALETTE
// UI_EVENT_MAP
// UI_WINDOW_OBJECT
// WOF_FLAGS
// WOAF_FLAGS
// WOS_STATUS
// UI_WINDOW_OBJECT::ZIL_INFO_REQUEST
// UI_HELP_CONTEXT
// Underline character information
// Border widths tor WOF_BORDER flag option
// UIW_WINDOW
// WNF_FLAGS
// UIW_WINDOW::ZIL_INFO_REQUEST
II UI_WINDOW_MANAGER
// U1W_B0RDER
// BDF_FLAGS
// UIW_PROMPT
// UIW_BUTTON
// BTF_FLAGS
// BTS_STATUS
// UIW_BUTTON::ZIL_INFO_REQUEST
// UIW_TITLE
```

```
// UIW_MAXIMIZE_BUTTON
// UIW_MINIMIZE_BUTTON
// UIW_ICON
// ICF_FLAGS
// UIW_ICON::ZIL_INFO_REQUEST
// UIW_POP_UP_MENU
// UIW_POP_UP_ITEM
// MNIF_FLAGS
// UIW_PULL_DOWN_MENU
// UIW_PULL_DOWN_ITEM
// UIW_SYSTEM_BUTTON
// SYF_FLAGS
// UIW_STRING
// STF_FLAGS
// UIW_DATE
// UIW_FORMATTED_STRING
// FMI_RESULT
// UIW_BIGNUM
// UIW_INTEGER
// UIW_REAL
// UIW_TIME
// UIW_TEXT
// UIW_GROUP
// UIW_VT_L1ST
// UIW_HZ_LIST
// UIW_COMBO_BOX
// UIW_COMBO_BOX::ZIL_INFO_REQUEST
// UIW_S PIN_CONTROL
// UIW_SCROLL_BAR
// sbFlags
// UIW_TOOL_BAR
// UIW_STATUS_BAR
// UIW_NOTEBOOK
// UIW_NOTEBOOK::ZIL_INFO_REQUEST
// UIW_TABLE
// UIW_TABLE::ZIL_INFO_REQUEST
// tblFlags
// thFlags
// UI_ERROR_SYSTEM
// UI_HELP_SYSTEM
// UI_APPLICATION
// ZAF_DIALOG_WINDOW
// ZIL_DIALOG_EVENT
// ZAF_MESSAGE_WINDOW
// UI_CONSTRAINT
// UI_CONSTRAINT::ZIL_INFO_REQUEST
// UI_ATTACHMENT
// ATCF_FLAGS
// UI_ATTACHMENT::ZIL_INFO_REQUEST
// UI_DIMENSION_CONSTRAINT
// DNCF_FLAG
// UI_DIMENSION_CONSTRAINT::ZIL_INFO_REQUEST
// UI_RELATTVE_CONSTRAINT
// RLCF_FLAG
// UI_RELATIVE_CONSTRAINT::ZIL_INFO_REQUEST
// UI_GF.OMETRY_MANAGER
// Message indexes for the help and error system windows.
// Version 3.6 compatibility
```

CLASS HIERARCHY




class UIW_WINDOW†
class UI_WINDOW_MANAGER
class UIW_COMBO_BOX
class UIW_GROUP
class UIW_HZ_LIST
class UIW_NOTEBOOK
class UIW_POP_UP_MENU
class UIW_PULL_DOWN_MENU
class UIW_SCROLL_BAR
class UIW_SPIN_CONTROL
class UIW_STATUS_BAR
class UIW_TABLE
class UIW_TABLE_HEADER
class UIW_TABLE_RECORD
class UIW_TEXT
class UIW_TOOL_BAR
class UIW_VT_LIST
class ZAF_DIALOG_WINDOW
class ZAF_MESSAGE_WINDOW
class ZIL_I18N
class ZIL_DECORATION
class ZIL_LANGUAGE
class ZIL_LOCALE
class UI_KEY
class UI_LIST
class UI_GEOMETRY_MANAGER†
class UI_EVENT_MANAGER
class UI_LIST_BLOCK
class UI_QUEUE_BLOCK
class UI_PATH†
class UI_REGION_LIST
class UI_BGI_DISPLAY†
class UI_GRAPHICS_DISPLAY†
class UI_MSC_DISPLAY†

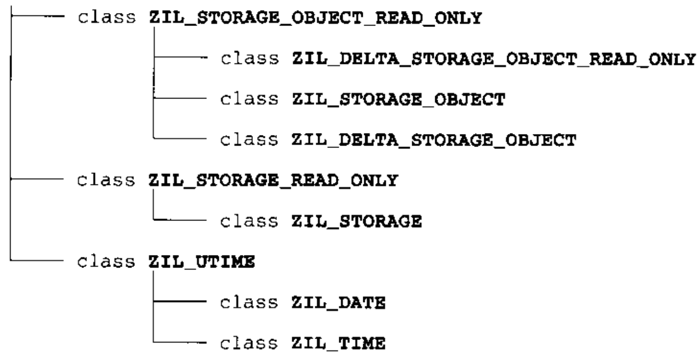
```
          |                 |---- class UI_TEXT_DISPLAY†
          |                 |
          |                 `---- class UI_WCC_DISPLAY†
          |
          |---- class UIW_WINDOW†
          |
          `---- class ZIL_I18N_MANAGER
                            |
                            |---- class ZIL_DECORATION_MANAGER
                            |
                            |---- class ZIL_LANGUAGE_MANAGER
                            |
                            `---- class ZIL_LOCALE_MANAGER

class UI_POSITION

class UI_REGION

class ZIL_BIT_VECTOR

class ZIL_INTERNATIONAL
          |
          |---- class UI_APPLICATION
          |
          |---- class UI_DISPLAY
          |                 |
          |                 |---- class UI_BGI_DISPLAY†
          |                 |
          |                 |---- class UI_GRAPHICS_DISPLAY†
          |                 |
          |                 |---- class UI_MACINTOSH_DISPLAY
          |                 |
          |                 |---- class UI_MSC_DISPLAY†
          |                 |
          |                 |---- class UI_MSWINDOWS_DISPLAY
          |                 |
          |                 |---- class UI_NEXTSTEP_DISPLAY
          |                 |
          |                 |---- class UI_OS2_DISPLAY
          |                 |
          |                 |---- class UI_PRINTER
          |                 |
          |                 |---- class UI_TEXT_DISPLAY†
          |                 |
          |                 |---- class UI_WCC_DISPLAY†
          |                 |
          |                 `---- class UI_XT_DISPLAY
          |
          |---- class UI_ERROR_STUB
          |                 |
          |                 `---- class UI_ERROR_SYSTEM
          |
          |---- class UI_HELP_STUB
          |                 |
          |                 `---- class UI_HELP_SYSTEM
          |
          |---- class UI_PATH†
          |
          |---- class UI_PATH_ELEMENT†
          |
          |---- class UI_WINDOW_OBJECT†
          |
          |---- class UID_CURSOR†
          |
          |---- class UID_KEYBOARD†
          |
          |---- class ZIL_BIGNUM
          |
```



class **ZIL_MAP_CHARS**

class **ZIL_MESSAGE**

class **ZIL_STORAGE_DIRECTORY**

struct **directoryEntry**

struct **UI_EVENT**

struct **UI_EVENT_MAP**

struct **UI_ITEM**

struct **UI_KEY**

struct **UI_PALETTE**

struct **UI_PALETTE_MAP**

struct **UI_POSITION**

struct **UI_REGION**

struct **UI_SCROLL_INFORMATION**

struct **ZIL_BITMAP_ELEMENT**

struct **ZIL_ICON_HANDLE**

struct **ZIL_LANGUAGE_ELEMENT**

struct **ZIL_LOCALE_ELEMENT**

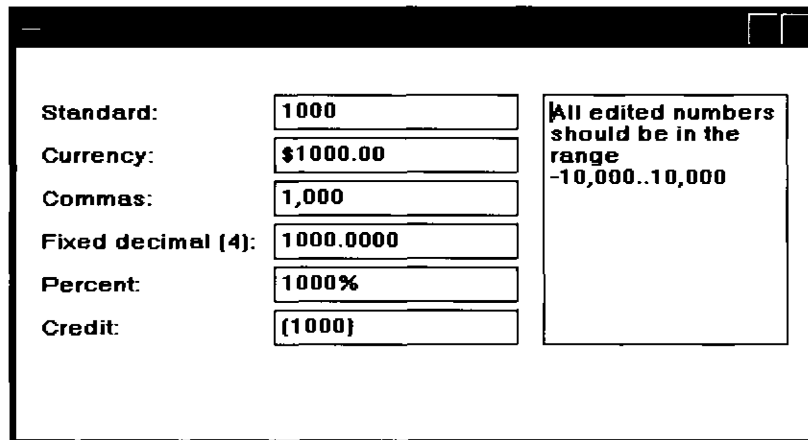
struct **ZIL_STATS_INFO**

struct **ZIL_TEXT_ELEMENT**

† - indicates multiple inheritance

CHAPTER 1 - UIW_BIGNUM

The UIW_BIGNUM class is used to display numeric information and to collect information, in numeric form, from an end-user. The UIW_BIGNUM object provides special formatting features (e.g., currency, credit, percent, etc.) as well as the capability to use large numbers with high precision. By default, UIW_BIGNUM allows up to 30 digits to the left of the decimal point and up to 8 digits to the right of the decimal point. The UIW_BIGNUM class is a high-level object that is used to interact with the end-user. It makes use of the ZIL_BIGNUM class, which is a low-level object that handles the details of numeric data manipulation. See "Chapter 49—ZIL_BIGNUM" of *Programmer's Reference Volume 1* for more information about the ZIL_BIGNUM class. The figure below shows a graphic implementation of a window with several variations of the UIW_BIGNUM class object:



The UIW_BIGNUM class is declared in UI_WIN.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_BIGNUM : public UIW_STRTNG
{
public:
    static ZIL_ICHAR _className[];
    static int defaultInitialized;
    NMF_FLAGS nmFlags;
#if defined(ZIL_3x_COMPAT)
    static NMF_FLAGS rangeFlags;
#endif

    UIW_BIGNUM(int left, int top, int width, ZIL_BIGNUM *value,
        const ZIL_ICHAR *range = ZIL_NULLP(ZIL_ICHAR),
        NMF_FLAGS nmFlags = NMF_NO_FLAGS,
        WOF_FLAGS woFlags = WOF_BORDER | WOF_AUTO_CLEAR,
        ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION) ) ;
```

```
    virtual ~UIW_BIGNUM(void);
    virtual ZIL_ICHAR *ClassName(void);
    ZIL_BIGNUM *DataGet(void);
    void DataSet(ZIL_BIGNUM *value);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);
    virtual int Validate(int processError = TRUE);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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 =
            Z1L_NULLP(Z1L_ST0RAGE_0BJECT_READ_0NLY),
        UI_ITEM *objectTable = ZIL_NULI,P (UI_1TEM) ,
        UI_ITEM *userTable = ZIL_NULLP(UI_ITEM));
    UIW_BIGNUM(const ZIL_ICHAR *name, ZIT,_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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

    void SetLanguage(const ZIL_ICHAR *languageName);

protected:
    ZIL_BIGNUM *number;
    ZIL_ICHAR *range;
    const ZIL_LANGUAGE *myLanguage;
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_BIGNUM class, *_className* is "UIW_BIGNUM."

- *defaultlnitialized* indicates if the default language strings for this object have been set up. The default strings are located in the file **LANG_DEF.CPP.** If *defaultlnitialized* is TRUE, the strings have been set up. Otherwise they have not been.

- *rangeFlags* are flags that define how the range values are interpreted. *rangeFlags* is set to NMF_NO_FLAGS by default.

- *nmFlags* are flags that define the operation of the UIW_BIGNUM class. A full description of the number flags is given in the UIW_BIGNUM constructor.

- *number* is a pointer to a ZIL_BIGNUM that is used to manage the low-level bignum information. If the WOF_NO_ALLOCATE_DATA flag is set, this member will simply point to the ZIL_BIGNUM value passed in.

- *range* is a string that specifies the range(s) of acceptable bignum values, *range* is a copy of the range that is passed to the constructor.

- *myLanguage* is the ZIL_LANGUAGE object that contains the string translations for this object.

UIW_BIGNUM::UIW_BIGNUM

Syntax

```
#include <ui_win.hpp>

UIW_BIGNUM(int left, int top, int width, ZIL_BIGNUM *value,
    const ZIL_ICHAR "range = ZIL_NULLP(ZIL_ICHAR),
    NMF_FLAGS nmFlags = NMF_NO_FLAGS,
    WOF_FLAGS woFlags = WOF_BORDER | WOF_AUTO_CLEAR,
    ZIL_USER_FUNCTION userFunction = ZIL_NULLP(ZIL_USER_FUNCTION));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_BIGNUM class object.

- $left_{in}$ and top_{in} is the starting position of the bignum field within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the bignum field. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value. The height of the bignum field is determined automatically by the UIW_BIGNUM object.

- $value_{in}$ is a pointer to a ZIL_BIGNUM object. Its value will be used as the initial value.

- $range_{in}$ is a string that specifies the valid numeric ranges. A range consists of a minimum value, a maximum value, and the values in between. For example, if a range of "1000.. 10000" is specified, the UIW_BIGNUM class object will only accept those numeric values that fall between 1,000 and 10,000, inclusive. Open-ended ranges can be specified by leaving the minimum or maximum value off. For example, a range of "500.." will allow all values that are 500 or greater. Multiple, disjoint ranges can be specified by separating the individual ranges with a slash (i.e. '/'). For example, "100.. 199/1000.." will accept all values from 100 to 199 and values of 1000 or greater. If *range* is NULL, any number within the absolute range is accepted. This string is copied by the UIW_BIGNUM class object to the *range* member variable.

- $nmFlags_{in}$ describes how the bignum should display and interpret the numeric information. The following flags (declared in **UI_GEN.HPP)** control the general presentation of a UIW_BIGNUM class object:

NMF_DECIMAL(*decimal*)—Displays the number with a specified number of decimal places. *decimal* is the number of decimal places to be displayed. The field can display from 0 to 8 decimal places.	10,000.00 43.45 $149.95
NMF_CURRENCY—Displays the number with the locale-specific currency symbol.	$10,000.00 DM100 £195
NMF_CREDIT—Displays the number with the locale-specific credit symbols whenever the number is negative.	(10000) (2350)

NMF_NO_FLAGS—Does not associate any special flags with the number object. This flag should not be used in conjunction with any other NMF flag. This is the default argument in the constructor.	10000 4345 149950
NMF_PERCENT—Displays the number with a percentage symbol.	100% 4.5% 10%
NMF_SCIENTIFIC—Causes the number to be formatted with scientific notation.	1.0e1 4.5e0 1.0e1

- *woFlags*$_{in}$ are flags (common to all window objects) that determine the general operation of the bignum object. The following flags (declared in UI_WIN.HPP) affect the operation of a UIW_BIGNUM class object:

  WOF_AUTO_CLEAR—Automatically marks the entire buffer if the end-user tabs to the field from another object. If the user then enters data (without first having pressed any movement or editing keys) the entire field will be replaced. This flag is set by default in the constructor.

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOF_INVALID—Sets the initial status of the field to be "invalid." Invalid entries fit in the absolute range determined by the object type but do not fulfill all the requirements specified by the program. For example, a bignum may initially be set to 200, but the final number, edited by the end-user, must be in the range "10..100." The initial number in this example fits the absolute range requirements of a UIW_BIGNUM class object but does not fit into the specified range. By denoting the field as invalid, you force the user to enter an acceptable value.

  WOF_JUSTIFY_CENTER—Center-justifies the data within the displayed field.

  WOF_JUSTIFY_RIGHT—Right-justifies the data within the displayed field.

WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is 1/10th the size of a normal cell by default.

WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

WOF_NO_FLAGS—Does not associate any special window flags with the object. Setting this flag left-justifies the data. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor edit the bignum information. Typically, the field will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

WOF_SUPPORT_OBJECT—Causes the object to be placed in the parent object's support list. The support list is reserved for objects that are not displayed as part of the user region of the window. Care should be used when setting this flag on an object that does not use it by default as undesirable effects may occur. This flag generally should not be used by the programmer.

WOF_UNANSWERED—Sets the initial status of the field to be "unanswered." An unanswered field is displayed as an empty field.

WOF_VIEW_ONLY—Prevents the object from being edited. However, the object may become current and the user may scroll through the data, mark it, and copy it.

userFunction$_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. The user function will be called by the library when:

1—the user moves onto the field,

2—the <ENTER> key is pressed while the field is current or, if the field is in a list, the mouse is clicked on it, or

3—the user moves to a different field in the window or to a different window.

Because the user function is called at these times, the programmer can do data validation or any other type of necessary operation. The definition of the *userFunction* is as follows:

EVENT_TYPE FunctionName(UI_WINDOW_OBJECT **object,*
UI_EVENT *&event,* EVENT_TYPE *ccode);*

returnValue$_{out}$ indicates if an error has occurred. *returnValue* should be 0 if the no error occurred. Otherwise, the programmer should call the error system with an appropriate error message and return -1.

object$_{in}$ is a pointer to the object for which the user function is being called. This argument must be typecast by the programmer if class-specific members need to be accessed.

event$_{in}$ is the run-time message passed to the object.

ccode$_{in}$ is the logical or system code that caused the user function to be called. This code (declared in **UI_EVT.HPP)** will be one of the following constant values:

L_SELECT—The <ENTER> key was pressed while the field was current or, if the field is in a list, the mouse was clicked on the field.

S_CURRENT—The object just received focus because the user moved to it from another field or window. This code is sent before any editing operations are permitted.

S_NON_CURRENT—The object just lost focus because the user moved to another field or window.

NOTE: If a user function is associated with the object, then **Validate(**) must be called explicitly from within *userFunction* if range checking is desired.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Create a window and add it to the window manager.
    ZIL_BIGNUM value(0);
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_BORDER
        + new UIW_TITLE(" Sample numbers ")
        + new UIW_PROMPT(2, 1, "Standard:")
        + new UIW_BIGNUM(12, 1, 20, &value, "0..10000")
        + new UIW_PROMPT(2, 2, "Currency:")
        + new UIW_BIGNUM(12, 2, 20, &value, "0..10000",
            NMF_CURRENCY | NMF_DECIMAL(2))
        + new UIW_PROMPT(2, 3, "Commas:")
        + new UIW_BIGNUM(12, 3, 20, &value, "0..10000", NMF_COMMAS);
    *windowManager + window;



    // The number fields are automatically destroyed when the window
    // is destroyed.
}
```

UIW_BIGNUM::~UIW_BIGNUM

Syntax

#include <ui_gen.hpp>

virtual ~UIW_BIGNUM(void);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_BIGNUM object.

Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_BIGNUM number("100");


    UIW_BIGNUM *bignum = new UIW_BIGNUM(1, 1, 20, &number);


    delete bignum;
}
```

UIW_BIGNUM::ClassName

Syntax

#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to *_className.*

UIW_BIGNUM::DataGet

Syntax

#include <ui_win.hpp>

ZIL_BIGNUM *DataGet(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function gets the current numeric information associated with the UIW_BIGNUM class object.

- *returnValue*$_{out}$ is a pointer to a ZIL_BIGNUM object containing the current bignum value.

Example

```
#include <ui_win.hpp>

EVENT_TYPE BignumUserFunction(UI_WINDOW_OBJECT *object, UI_EVENT &, EVENT_TYPE
ccode)
{
    if (ccode != S_NON_CURRENT)
        return (ccode);

    // Do specific validation.
    ZIL_BIGNUM *bignum = ((UIW_BIGNUM *)object)->DataGet();

    // Call the default Validate function to check for valid bignum.
    int valid = object->Validate(TRUE);

    // Call error system if the bignum is larger than maximum value,
    extern ZIL_BIGNUM _maxValue;
    if (valid == NMI_OK && *bignum > _maxValue)
    {
        valid = NMI_OUT_OF_RANGE;
        char bignumString[64];
        _maxValue.Export(bignumString, 64, NMF_NO_FLAGS);
        object->errorSystem->ReportError(object->windowManager, WOS_NO_STATUS,
            "The bignum must be less than %s.", bignumString);
    }

    // Return error status,
    if (valid == NMI_OK)
        return (0) ;
    else
        return (-1);

void ExampleFunction1 (UI_WINDOW_MANAGF.R *wi ndowMar.ager)
{
    ZIL_BIGNUM value1, value2;
    UIW_WINDOW *window = UIW_WINDOW::Generic (0, 0, 45, 8, "Bignum Window");
    *window
        + new UIW_PROMPT(2, 1, "Initial value:")
        + new UIW_BIGNUM(12, 1, 20, &value1, NULL, NMF_NO_FLAGS,
            WOF_BORDER | WOF_AUTO_CLEAR, BignumUserFunction)
```

```
        + new UIW_PROMPT(2, 3, "Ending value:")
        + new UIW_BIGNUM(12, 3, 20, &value2, NULL, NMF_NO_FLAGS,
            WOF_BORDER | WOF_AUTO_CLEAR, BignumUserFunction);
    *windowManager + window;
}
```

UIW_BIGNUM::DataSet

Syntax

#include <ui_win.hpp>

void DataSet(ZIL_BIGNUM **value);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function assigns a new value to the UIW_BIGNUM object and redisplays the field. If no value is passed in (i.e., *value* is NULL), the field will be redrawn.

- $value_{in}$ is a pointer to the new value. If the WOF_NO_ALLOCATE_DATA flag is set, this argument must be a ZIL_BIGNUM, allocated by the programmer, that is not destroyed until the UIW_BIGNUM class object is destroyed. Otherwise, the information associated with this argument is copied by the UIW_BIGNUM class object. Care should be taken to only reset a value that is the same type as the original value. If this argument is NULL, no numeric information is changed, but the number field is redisplayed.

Example

```
•tinclude <ui_win.hpp>

Examp]eFuncc.ionl (UT_WINDOW_MANAGER *windowManager)
{
    // Manually add a number field to the window.
    ZIL_BIGNUM value(0);
    UIW_BIGNUM *numberField;
    UIW_WTNDOW *window - new U1W_WIND0W(0, 0, 40, 10);
    *window
```

```
        + new UIW_BORDER
        + new UIW_TITLE("Sample numbers")
        + new UIW_PROMPT(2, 1, "Standard: ")
        + (numberField = new UIW_BIGNUM(22, 1, 20, &value, "0..10000"));
    *windowManager + window;



    // Reset the numeric information for the field.
    value->Import(100);
    numberField->DataSet(&value);


}
```

UIW_BIGNUM::Event

Syntax

#include <ui_win.hpp>

virtual EVENT_TYPE Event(const UI_EVENT *&.event);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the bignum object. It is declared virtual so that any derived bignum class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the bignum object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event( ):**

L_SELECT—Indicates that the object has been selected. The selection may be the result of a mouse click or a keyboard action.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another field or window.

All other events are passed by **Event()** to **UIW_STRING::Event( )** for processing.

UIW_BIGNUM::Information

Syntax

#include <ui_win.hpp>

void *Information(ZIL_INFO_REQUEST *request,* void **data,*
ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc objects and programmer functions to get or modify specified information about an object.

- $returnValue_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

request$_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the bignum:

I_CLEAR_FLAGS—Clears the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be cleared, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to clear the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_-OBJECT. If the NMF_FLAGS are to be cleared, *objectID* should be ID_-BIGNUM. This allows the object to process the request at the proper level. This request only clears those flags that are passed in; it does not simply clear the entire field.

I_DECREMENT_VALUE—Decrements the bignum's value. If this message is sent, *data* must be a pointer to an integer. The bignum object's value will be decremented by the value of *data.* The bignum will not be modified if the new value is not within the specified range.

I_GET_FLAGS—Requests the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to obtain the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the NMF_FLAGS are desired, *objectID* should be ID_BIGNUM. This allows the object to process the request at the proper level.

I_GET_VALUE—Returns the *value* associated with the bignum. If this message is sent, *data* must be a pointer to a variable of type ZIL_BIGNUM where the bignum's value will be copied.

I_INCREMENT_VALUE—Increments the bignum's value. If this message is sent, *data* must be a pointer to an integer. The bignum object's value will be incremented by the value of *data.* The bignum will not be modified if the new value is not within the specified range.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_FLAGS—Sets the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be set, and *objectID* should indicate the type of object with which the

flags are associated. For example, if the programmer wishes to set the WOF_-FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the NMF_-FLAGS are to be set, *objectID* should be ID_BIGNUM. This allows the object to process the request at the proper level. This request only sets those flags that are passed in; it does not clear any flags that are already set.

I_SET_VALUE—Sets the *value* associated with the bignum. If this message is sent, *data* must be a pointer to a variable of type ZIL_BIGNUM that contains the bignum's new value.

All other requests are passed by **Information**() to **UIW_STRING::Information()** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information**() function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID*, the object will attempt to interpret the request with the *objectID* of the actual object type.

UIW_BIGNUM::SetLanguage

Syntax

```
#include <ui_win.hpp>

void SetLanguage(const ZIL_ICHAR *languageName);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function sets the language to be used by the object. The string translations for the object will be loaded and the object's *myLanguage* member will be updated to point to the new ZIL_LANGUAGE object. By default, the object uses the language identified in the **LANG_DEF.CPP** file, which compiles into the library. (If a different default language is desired, simply copy a **LANG_<ISO>.CPP** file from the OpenZinc\SOURCE\-INTL directory to the \OpenZinc\SOURCE directory, and rename it to **LANG_DEF.CPP** before compiling the library.) The language translations are loaded from the **I18N.DAT** file, so it must be shipped with your application.

- $languageName_{in}$ is the two-letter ISO language code identifying which language the object should use.

UIW_BIGNUM::Validate

Syntax

#include <ui_win.hpp>

virtual int Validate(int *processError* = TRUE);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function is used to validate objects. When an object receives the S_CURRENT or S_NON_CURRENT messages, it calls **Validate( )** to check if the value entered is valid. However, if a user function is associated with the object, **Validate( )** must be called explicitly from the user function if range checking is desired. The value is invalid if it is not within the absolute range of the object or if it is not within a range specified by the *range* member variable.

- $returnValue_{out}$ indicates the result of the validation. The possible values for *returnValue* are:

NMI_GREATER_THAN_RANGE—The number entered was greater than the maximum value of a negatively open-ended range.

NMI_INVALID—The number was entered in an incorrect format.

NMI_LESS_THAN_RANGE—The number entered was less than the minimum value of a positively open-ended range.

NMI_OK—The number was entered in a correct format and within the valid ranges.

NMI_OUT_OF_RANGE—The number was not within the valid range for numbers or was not within the specified *range.*

- *processError*$_{in}$ determines whether **Validate( )** should call **UI_ERROR_SYSTEM::-ReportError()** if an error occurs. If *processError* is TRUE, **ReportError( )** is called. Otherwise, the error system is not called.

Example

```
#include <ui_win.hpp>

EVENT_TYPE BignumUserFunction(UI_WINDOW_OBJECT *object, UI_EVENT &,
    EVENT_TYPE ccode)
{
    if (ccode != S_NON_CURRENT)
        return (ccode);

    // Do specific validation.
    ZIL_BIGNUM *bignum = ((UIW_BIGNUM *)object)->DataGet();

    // Call the default Validate function to check for valid bignum.
    int valid = object->Validate(TRUE);

    // Call error system if the bignum is larger than maximum value,
    extern ZIL_BIGNUM _maxValue;
    if (valid == NMI_OK && *bignum < _maxValue)
    {
        valid = NMI_OUT_OF_RANGE;
        char bignumString[64];
        _maxValue.Export(bignumString, 64, NMF_NO_FLAGS);
        object->errorSystem->ReportError(object->windowManager,WOS_NO_STATUS,
            "The bignum must be less than %s.", bignumString);
    }
    // Return error status,
    if (valid == NMI_OK)
        return (0);
    else
        return (-1);
}

void ExampleFunction1(UI_WINDOW_MANAGER *windowManager)
{
    UIW_WINDOW *window = UIW_WINDOW::Generic(0, 0, 45, 8, "National Debt");
    *window
```

```
		+ new UIW_PROMPT(2, 1, "Initial value:")
		+ new UIW_BIGNUM(12, 1, 20, &ZIL_BIGNUM(), NULL, NMF_NO_FLAGS,
			WOF_BORDER | WOF_AUTO_CLEAR, BignumUserFunction)
		+ new UIW_PROMPT(2, 3, "Ending value:")
		+ new UIW_BIGNUM(12, 3, 20, &ZIL_BIGNUM(), NULL, NMF_NO_FLAGS,
			WOF_BORDER | WOF_AUTO_CLEAR, BignumUserFunction);
	*windowManager + window;
}
```

Storage Members

This section describes those class members that are used for storage purposes.

UIW_BIGNUM::UIW_BIGNUM

Syntax

#include <ui_win.hpp>

UIW_BIGNUM(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));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_BIGNUM by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a bignum is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW BIGNUM::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced function is used to load a UIW_BIGNUM from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UIJWIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:.userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_BIGNUM::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-

OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_BIGNUM::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New()** function.

- *returnValue*$_{out}$ is a pointer to the object's **New( )** function.

UIW_BIGNUM::Store

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

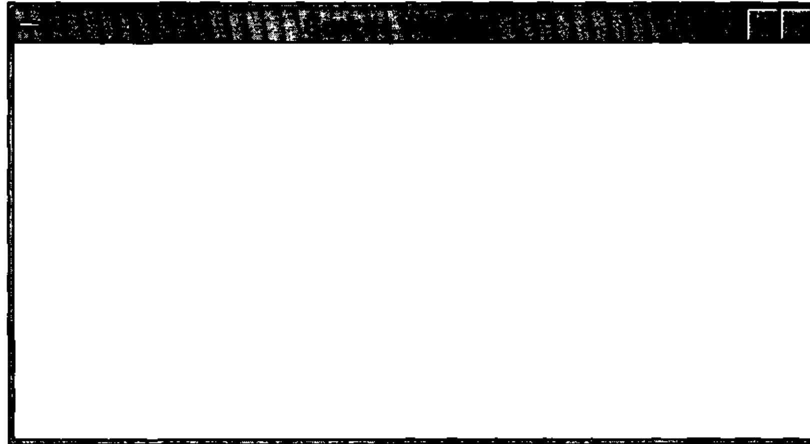
- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 2 - UIW_BORDER

The UIW_BORDER class is used to draw a border around a window in graphics modes only. The figure below shows a graphic implementation of a window which contains a UIW_BORDER class object (the outer-most region of the window):



The UIW_BORDER class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_BORDFR : public UI_WINDOW_OBJECT
{
public:
    static int width;
    static ZIL_ICHAR _className[];
    BDF_FLAGS bdFlags;

    UIW_BORDER(BDF_FLAGS bdFlags = BDF_NO_FLAGS);
    virtual ~UIW_BORDER(void);
    virtual ZIL_ICHAR *ClassName(void);
    int DataGet(void);
    void DataSet(int width);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_BORDER(const ZIL_ICHAR *name, ZIL_STORAGE_READ_ONLY *file,
        ZIL_ST'ORAGE_OBJECT_READ_ONLY * object,
        UI_ITEM *objectTable = ZIL_NULLP (UI_ITEM) ,
        UI_ITEM *userTabie - 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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);

#endif

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

General Members

This section describes those members that are used for general purposes.

- *width* is the default width used when an application is running in graphics mode. The pre-defined value for *width* is 4 pixels. A greater value increases the width of the border displayed on the screen.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_BORDER class, *_className* is "UIW_BORDER."

- *bdFlags* are flags that define the operation of the UIW_BORDER class. A full description of the border flags is given in the UIW_BORDER constructor.

UIW_BORDER::UIW_BORDER

Syntax

#include <ui_win.hpp>

UIW_BORDER(BDF_FLAGS bdFlags = BDF_NO_FLAGS);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_BORDER object. The border object always occupies the outer-most space available in the parent window. To ensure that the border is drawn around the whole window, it must be created as the window's first object. The following example shows the correct and incorrect order of border creation:

```
1)   // CORRECT construction order.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_BORDER
        + new UIW_MAX1MIZE_BUTT0N
        + new UIW_MINIMIZE_BUTTON
        + new UIW_SYSTEM_BUTTON
        + new UIW_TITLE("Window 1")
```

```
2)   // INCORRECT construction order.
    UIW_WINDOW *window = new UIW_WTNDOW(0, 0, 40, 10);
    *window
        * new UIW_MAXIMIZE_BUTTON
        + new UIW_MINIMIZE_BUTTON
        + new UIW_SYSTEM_BUTTON
        + new UIW_TITLE("Window 1")
        + new UIW_BORDER
```

The UIW_BORDER class should not be confused with the WOF_BORDER flag. The UIW_BORDER class is attached as an object to a parent window and allows sizing with the mouse (unless the window has the WOAF_NO_SIZE flag set). The UIW_BORDER class also typically has a more elaborate, 3-dimensional appearance. The WOF_BORDER flag is not an object and only draws a thin, solid line around the object. If operating in text mode, neither the UIW_BORDER or the WOF_BORDER may draw, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles.

NOTE: When a UIW_BORDER is added to a window, it is put into the window's support list. For more information regarding the support list, see the section on *UIW_-WINDOW:-.support* in "Chapter 33—UIW_WINDOW" of this manual.

- *bdFlags*$_{in}$ describes how the border should operate. The following flags (declared in **UI_WIN.HPP)** control the general presentation and operation of a UIW_BORDER object:

    BDF_NO_FLAGS—Does not associate any special flags with the UIW_-BORDER class object. This flag is set by default in the constructor.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Create a new window and attach it to the window manager.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_BORDER
        + new UIW_MAXIMIZE_BUTTON
        + new UIW_MINIMIZE_BUTTON
        + new UIW_SYSTEM_BUTTON
        + new UIW_TITLE("Window 1");
    *windowManager + window;

    // The border will automatically be destroyed when the window
    // is destroyed.
```

UIW_BORDER::ClassName

Syntax

#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{in}$ is a pointer to *_className.*

UIW_BORDER::DataGet

Syntax

#include <ui_win.hpp>

int DataGet(void);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function returns the width of the border object.

- *returnValue*$_{out}$ is the width of the border.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_BORDER *border)
{
    int width = border->DataGet();
```

UIW_BORDER::DataSet

Syntax

#include <ui_win.hpp>

void DataSet(int *width);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function sets the width of the border object. Since *UIW_BORDER::width* is static, border objects created after a call to **DataSet()** will all be made with the new border size.

- *width*$_{in}$ is the new width of the border.

Example

```
#include <ui_win.hpp>

ExampleFunction1(UIW_BORDER *border)
{
    int newWidth = 6;
    border->DataSet(newWidth);
}
```

UIW_BORDER::DrawItem

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual advanced function is used to draw the object. If the WOS_OWNERDRAW status is set for the object, this function will be called when drawing the border. This allows the programmer to derive a new class from UIW_BORDER and handle the drawing of the border, if desired.

- $returnValue_{out}$ is a response based on the success of the function call. If the object is drawn the function returns a non-zero value. If the object is not drawn, 0 is returned.

- $event_{in}$ contains the run-time message that caused the object to be redrawn. *event.region* contains the region in need of updating. The following logical events may be sent to the **DrawItem()** function:

  S_CURRENT, S_NON_CURRENT, S_DISPLAY_ACTIVE and **S_DISPLAY_INACTIVE**—Messages that cause the object to be redrawn.

  NOTE: The **DrawItem()** function draws the border in DOS only. In all other environments, the operating system is responsible for drawing the border.

- $ccode_{in}$ contains the logical interpretation of *event.*

UIW_BORDER::Event

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function processes run-time messages sent to a border object. It is declared virtual so that any derived border class can override its default operation.

- *returnValue*$_{out}$ is a response indicating how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from the *event* reference variable. If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the border object. The type of operation depends on the interpretation of the event. The following logical events are processed by Event():

  L_BEGIN_SELECT—If the parent window can be sized, this message causes the border to begin the size operation. When the user releases the mouse button, the S_CHANGED message is sent to the window, indicating that the window has changed size. The type of size operation is determined by the overlap of *event.-position* within the border.

  L_VIEW—Changes the mouse pointer (if any) shown on the screen to reflect the way in which the window may be sized. For instance, if the user positions the mouse on the top part of a window's border the mouse pointer is changed to indicate that vertical sizing can take place.

  S_CHANGED—Causes the border to change its size according to the size of its parent object. This message is received after the window has been sized.

  S_CREATE—Causes the border to create itself according to the size of its parent object. The border always occupies the outermost region of its parent object. The border is not shown until a display message (S_DISPLAY_-INACTIVE or S_DISPLAY_ACTIVE) is received by the border object.

  S_DISPLAY_ACTIVE and S_DISPLAY_INACTIVE—Cause the border to be re-drawn if *event.region* overlaps any part of the border.

  All other events are passed by Event( ) to UI_WINDOW_OBJECT::Event( ) for processing.

  NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own

messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event( )** function.

UIW_BORDER::Information

Syntax

```
#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST request, void *data,
    ZIL_OBJECTID objectID = ID_DEFAULT);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc objects and programmer functions to get or modify specified information about an object. It is declared virtual so that any derived border class can override its default operation. **Information( )** does not process any messages; it simply passes them to **UI_WINDOW_OBJECT::Information().**

Storage Members

This section describes those class members that are used for storage purposes.

UIW_BORDER::UIW_BORDER

Syntax

```
#include <ui_win.hpp>

UIW_BORDER(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));
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced constructor creates a new UIW_BORDER by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a border is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library

will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_BORDER::Load

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced function is used to load a UIW_BORDER from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:-.objectTable* in "Chapter 43—UI_WIN-

DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_BORDER::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume* 7.

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume* |. If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW BORDER::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New()** function.

- *returnValue*$_{in}$ is a pointer to the object's **New()** function.

UIW_BORDER::Store

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

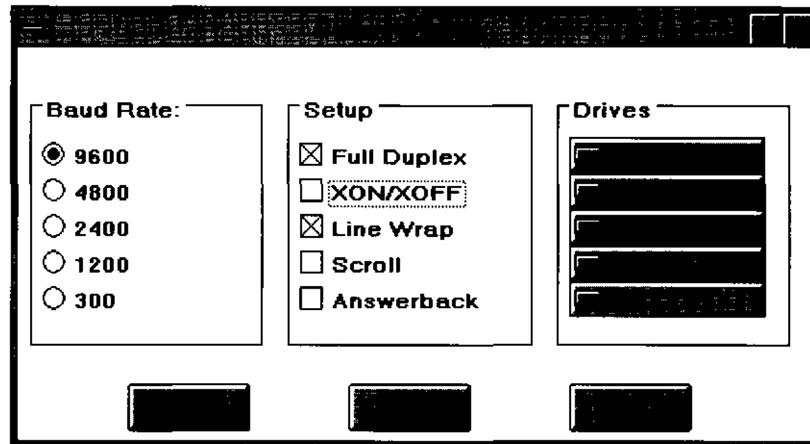
- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see

the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 3 - UIW_BUTTON

The UIW_BUTTON class is used to initiate an action or to obtain, from the end-user, one or more selections from a group of related options. The UIW_BUTTON class can create check boxes, radio buttons and normal push buttons, including default buttons. Push buttons can be either three-dimensional or flat. Check boxes and radio buttons are typically used to present a list of options to the end-user and to allow the end-user to make the desired selections. Normal push buttons are typically used to allow the end-user to indicate to the program that the action represented by that button should be performed. A default button is selected if the end-user presses the <ENTER> key anywhere on the window and the current field cannot process the event. The figure below shows the graphical implementation of UIW_BUTTON objects:



The UIW_BUTTON class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_BUTTON : public UI_WINDOW_OBJECT
{
public:
    static ZIL_ICHAR _className[];
    static int defaultInitialized;
    BTF_FLAGS btFlags;
    EVENT_TYPE value;

    UIW_BUTTON(int left, int top, int width, ZIL_ICHAR *text,
        BTF_FLAGS btFlags = BTF_NO_TOGGLE | BTF_AUTO_SIZE,
        WOF_FLAGS woFlags = WOF_JUSTIFY_CENTER,
        ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION),
        EVENT_TYPE value = 0, ZIL_ICHAR *bitmapName = ZIL_NULLP(ZIL_ICHAR));
    virtual ~UIW_BUTTON(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    ZIL_ICHAR *DataGet(int stripText = FALSE);
```

```
    void DataSet(ZIL_ICHAR *text);
    virtual void "Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);
    static EVENT_TYPE Message(UI_WINDOW_OBJECT *object, UI_EVENT &event,
        EVENT_TYPE ccode);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    static UI_WINDOW_OBJECT *New(const ZIL_ICHAR *name,
        ZTI,_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));
    UTW_BUTTON(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
#if defined (ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UT_ITEM *userTable);
#endif

    void SetDecorations(const ZIL_ICHAR *decorationName);

protected:
    BTS_STATUS btStatus;
    int depth;
    ZIL_ICHAR *text;
    ZIL_ICHAR *bitmapName;
    int bitmapWidth;
    int bitmapHeight;
    ZIL_UINT8 *bitmapArray;
    ZIL_BITMAP_HANDLE colorBitmap, monoBitmap;
#if defined(ZIL_XT)
    Pixmap pixmap;
#endif

    const ZIL_DECORATION *myDecorations;

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

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_BUTTON class, *_className* is "UIW_BUTTON."

- *defaultlnitialized* indicates if the default decorations (i.e., images) for this object have been set up. The default decorations are located in the file **IMG_DEF.CPP.** If

defaultlnitialized is TRUE, the decorations have been set up. Otherwise they have not been.

- *btFlags* are flags that define the operation of the UIW_BUTTON class. A full description of the button flags is given in the UIW_BUTTON constructor.

- *value* is an event that will be placed on the event queue if the button has the BTF_-SEND_MESSAGE flag set and the end-user selects the button, *value* can also be used as an identifier to distinguish between several buttons in a common user function. For example, the programmer could associate the value 0 with an "OK" button and a value of 1 with a "Cancel" button. This allows the programmer to define one user function that can determine the action to take based on the button's value.

- *btStatus* are status flags that indicate the current status of a button. The following internal status flags may be set for a UIW_BUTTON object:

  BTS_DEFAULT—The button is the default button on the window and appears with a thick border.

  BTS_DEPRESSED—The button is pressed down and appears depressed (where applicable).

  BTS_NO_STATUS—The button is in a normal state.

- *depth* specifies how three-dimensional the button appears. The greater the value of *depth,* the more the button will appear to recess into the screen or pop out of the screen. The following values are used for *depth* by default:

  0—The button is shown flat, *depth* will be 0 if the BTF_NO_3D flag is set.

  2—The button will have a three-dimensional appearance.

- *text* is the text that is shown on the button (including all trailing and leading spaces or check marks, etc.).

- *bitmapName* is the bitmap's name in a OpenZinc .DAT file or an operating system resource file. *bitmapName* is used if the bitmap is to be read from a OpenZinc .DAT file or an operating system resource file.

- *bitmapWidth* is the pixel width of the button's bitmap.

- *bitmapHeight* is the pixel height of the button's bitmap.

- *bitmapArray* is a pointer to an array of ZIL_UINT8 composing the bitmap to be displayed on the button.

- *colorBitmap* is an ZIL_BITMAP_HANDLE structure that is specific to the native environment. *colorBitmap* is the bitmap image to be displayed on the button.

- *monoBitmap* is an ZILJ3ITMAPJIANDLE structure that is specific to the native environment. *monoBitmap* is a mask that specifies which pixels of the *colorBitmap* to draw and which ones to ignore, thus creating a transparent area in the bitmap.

- *pixmap* is a pointer to the image to be displayed on the button. If the button is to contain both text and an image, then *pixmap* will contain the image and the text. This member is specific to Motif.

- *myDecorations* is the ZIL_DECORATION object that contains the images for this object.

UIW_BUTTON::UIW_BUTTON

Syntax

```
#include <ui_win.hpp>

UIW_BUTTON(int left, int top, int width, ZIL_ICHAR *text,
    BTF_FLAGS btFlags = BTF_NO_TOGGLE | BTF_AUTO_SIZE,
    WOF_FLAGS woFlags = WOF_JUSTIFY_CENTER,
    ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION),
    EVENT_TYPE value = 0, ZIL_ICHAR *bitmapName = ZIL_NULLP(ZIL_ICHAR));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_BUTTON class object.

- $left_{in}$ and top_{in} is the starting position of the button field within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the button. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value. The height of the button is determined automatically by the UIW_BUTTON object.

- $text_{in}$ is the text that is shown on the button. A hotkey for the button may be specified by inserting the '&' character into the string before the desired hotkey character. For example, if the string "Exit" is to be displayed and 'x' is to be the hotkey, the string should be entered as "E&xit." The '&' will not be displayed, but will cause the hotkey character to be drawn appropriately. If an '&' is required in the text that is displayed, place two '&' characters in the string (e.g., "A && B" will display as "A & B" and the button will not have a hotkey). In those environments that don't support hotkeys on buttons (e.g., Macintosh, NEXTSTEP) the '&' character will not be displayed and will have no effect. This string is copied by the UIW_BUTTON class unless the WOF_NO_ALLOCATE_DATA flag is set. If this flag is set, *text* must be space, allocated by the programmer, that is not deleted until the UIW_BUTTON object has been deleted.

- $btFlags_{in}$ are flags that define the operation of the UIW_BUTTON class. The following flags (declared in **UI_WIN.HPP)** control the general presentation of a UIW_BUTTON class object:

  BTF_AUTO_SIZE—Automatically computes the run-time height of the button. The button typically will appear slightly taller than a button without this flag set. In text mode the button may not be affected. This flag is set by default in the constructor.

  BTF_CHECK_BOX—Creates the button as a check box. A check box is typically used in conjunction with other check boxes to allow the end-user to select from a set of options. A check box can be toggled on and off and is not mutually exclusive of other check boxes (i.e., more than one related check box can be selected at a time). In addition to the text the check box has a small image that toggles on and off as the button is selected and de-selected. Check boxes should be added to a UIW_GROUP or other grouping object. The group

must have the WNF_SELECT_MULTIPLE flag set to allow the full check box functionality.

BTF_DEFAULT_BUTTON—Creates the button as the default button on the window. The default button will be selected if the end-user hits <ENTER> anywhere on the window and the current object cannot process the event. The default button will have a dark border to distinguish it as the default button. Only one button can be the default button on a window. If another button becomes current, that button will act as if it is the default button until it is no longer current.

BTF_DOUBLE_CLICK—Causes the button to initiate its action when the end-user double-clicks on the button. A double-click consists of two down-click, up-click actions occurring within a period of time specified by *UI_WINDOW_-OBJECT::doubleClickRate.* If this flag is set, the user function will be called with a ccode of L_SELECT on the first click and L_DOUBLE_CLICK on the second click. This allows different actions to be performed on a single- and double-click.

BTF_DOWN_CLICK—Causes the button to initiate its action on a button down-click, rather than on a down-click and release action.

BTF_NO_FLAGS—Does not associate any special flags with the UIW_-BUTTON class object. In this case the button requires a down and up click from the mouse to complete an action. The button will also appear approximately one cell tall and will toggle when selected or de-selected.

BTF_NO_TOGGLE—Does not toggle the button's appearance when it is selected or de-selected. If this flag is not set, a push button will appear different if selected. In most environments the button will appear flat if it is selected or will pop out of the screen if it is not selected. On NEXTSTEP the button will change color to indicate it is selected. This flag is set by default in the constructor.

BTF_NO_3D—Causes the button to be displayed without a three-dimensional appearance.

BTF_RADIO_BUTTON—Creates the button as a radio button. A radio button is typically used in conjunction with other radio buttons to allow the end-user to select from a set of options. Radio buttons are mutually exclusive (i.e., only one radio button in a group of related radio buttons can be selected at a time) so that when a radio button is toggled on the currently selected radio button is toggled

off. In addition to the text the radio button has a small image that toggles on and off as the button is selected and de-selected. Radio buttons should be added to a UIW_GROUP or other grouping object.

BTF_REPEAT—Causes the button's action to be repeatedly performed if the button is held down. Most operating systems determine the repeat rate, but *UI_-WINDOWJOBJECT::repeatRate* will affect how often the action occurs in DOS.

BTF_SEND_MESSAGE—Causes an event to be created and put on the Event Manager queue when the button is selected. The button's *value* is placed in the EVENT_TYPE field of the posted event.

BTF_STATIC_BITMAPARRAY—Causes the bitmap array that is used for the image on a bitmap button to not be deleted. By default, when a bitmap button is created, the bitmap is converted to a native storage structure and the bitmap is deleted. If the bitmap should not be deleted after this conversion is performed (e.g., if the same bitmap image is to be used for multiple objects), then the BTF_STATIC_BITMAPARRAY flag should be set.

- $woFlags_{in}$ are flags (common to all window objects) that determine the general operation of the button object. The following flags (declared in UI_WIN.HPP) control the general presentation of, and interaction with, a UIW_BUTTON class object:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOF_JUSTIFY_CENTER—Center-justifies the text within the displayed button.

  WOF_JUSTIFY_RIGHT—Right-justifies the text within the displayed button.

  WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is 1/1 Oth the size of a normal cell by default.

WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

WOF_NO_FLAGS—Does not associate any special window flags with the object. Setting this flag left-justifies the data. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor select the button. Typically, the object will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

$userFunction_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. The user function will be called by the library when:

1—the user moves onto the field,

2—the <ENTER> key is pressed while the button is current or the button is the default button, or

3—the mouse is clicked on the object or, if the button has the BTF_-DOUBLE_CLICK flag set, the mouse is double-clicked on the button, or

4—the user moves to a different field in the window or to a different window.

Because the user function is called at these times, the programmer can do data validation or any other type of necessary operation. The definition of the *userFunction* is as follows:

```
EVENT_TYPE FunctionName(UI_WINDOW_OBJECT *object,
    UI_EVENT &event, EVENT_TYPE ccode);
```

$returnValue_{out}$ indicates if an error has occurred. *returnValue* should be 0 if the no error occurred. Otherwise, the programmer should call the error system with an appropriate error message and return -1.

$object_{in}$ is a pointer to the object for which the user function is being called. This argument must be typecast by the programmer if class-specific members need to be accessed.

$event_{in}$ is the run-time message passed to the object.

$ccode_{in}$ is the logical or system code that caused the user function to be called. This code (declared in UI_EVT.HPP) will be one of the following constant values:

L_DOUBLE_CLICK—The button has the BTF_DOUBLE_CLICK flag set and the user double-clicked on the button.

L_SELECT—The <ENTER> key was pressed while the field was current or the button is the default button, or the button was clicked on with the mouse.

S_CURRENT—The object just received focus because the user moved to it from another field or window.

S_NON_CURRENT—The object just lost focus because the user moved to another field or window.

- $value_{in}$ is an event that will be placed on the event queue if the button has the BTF_SEND_MESSAGE flag set and the end-user selects the button, *value* can also be used as an identifier to distinguish between several buttons in a common user function. For example, the programmer could associate the value 0 with an "OK" button and a value of 1 with a "Cancel" button. This allows the programmer to define one user function that can determine the action to take based on the button's value.

- $bitmapName_{in}$ is the bitmap's name in a OpenZinc .DAT file or an operating system resource file. *bitmapName* is used if the bitmap is to be read from a OpenZinc .DAT file or an operating system resource file.

Example

```
#include <ui_win.hpp>

const EVENT_TYPE FILE_EXIT  = 10001;
const EVENT_TYPE FILE_CANCEL= 10002;
const EVENT_TYPE FILE_HELP  = 10003;

static EVENT_TYPE FileControl (UI_WTNDOW_OBJECT *data, U1_EVENT &event,
    EVENT_TYPE ccode)
```

```
{
    // Switch on the event value (which is the button value).
    UIW_BUTTON *button = (UIW_BUTTON *)data;
    switch (button->value)
    {
    case FILE_EXIT:
        // Exit the application,
    EVENT_TYPE  =    L_EXIT;
        button->eventManager->Put(event);
        break;
    case FILE_CANCEL:
        // Close the file window,
        EVENT_TYPE =  S_CLOSE;
        button->eventManager->Put(event);
        break;

    case FILE_HELP:
        // Get help on the file program.
        _helpSystem->DisplayHelp(button->windowManager, HELP_FILE);
        break;
    }
    return ccode;
}

ExampleFunction1(U1_WINDOW_MANAGER *windowManager)
{
    // Create a window with buttons.
    UIW_WINDOW *window = UIW_WINDOW::Generic(0, 0, 50, 10, "Window");
    *window
        + new UIW_BUTTON(7, 6, 10, "E~xit", BTF_NO_TOGGLE | BTF_AUTO_SIZE,
            WOF_BOHDER | WOF_JUSTIFY_CENTER, FileControl, FILE_EXIT)
        + new UIW_BUTTON(20, 6, 10, ""Cancel", BTF_NO_TOGGLE | BTF_AUTO_SIZE,
            WOF_BORDER | WOF_JUSTIFY_CENTER, FileControl, FILE_CANCEL)
        + new UIW_BUTTON(33, 6, 10, ""Help", BTF_NO_TOGGLE | BTF_AUTO_SIZE,
            WOF_BORDER | WOF_JUSTIFY_CENTER, FileControl, F1LE_HELP,
            "HELP_BITMAP");
    *windowManager + window;



    // The buttons will automatically be destroyed when the window
    // is destroyed.
}
```

UIW_BUTTON::~UIW_BUTTON

Syntax

#include <ui_win.hpp>

virtual ~UIW_BUTTON(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_BUTTON object.

UIW_BUTTON::ClassName

Syntax

#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- $returnValue_{out}$ is a pointer to _className.

UIW_BUTTON::DataGet

Syntax

#include <ui_win.hpp>

ZIL_ICHAR *DataGet(int *stripText* = FALSE);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function gets the text associated with the button object.

- *returnValue*$_{out}$ is a pointer to the text associated with the button.

- *stripText*$_{in}$ indicates if the function should strip out any leading or trailing spaces that may be in the string or the hotkey character (i.e., '&') if one occurs in the string. If *stripText* is TRUE, the text will be stripped of these characters. Otherwise the characters will be in the returned string. *stripText* is FALSE by default if no other value is specified.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_BUTTON *button)
{
    ZIL_ICHAR *text = button->DataGet();


}
```

UIW_BUTTON::DataSet

Syntax

#include <ui_win.hpp>

void DataSet(ZIL_ICHAR **text);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function assigns new text to the button and redisplays the button. If no text is passed in (i.e., *text* is NULL), the button will be redrawn.

- $text_{in}$ is a pointer to the new text information to be displayed on the button. If the WOF_NO_ALLOCATE_DATA flag is set, *text* must be a string, allocated by the programmer, that is not destroyed until the UIW_BUTTON class object is destroyed. Otherwise, the information associated with this argument is copied by the UIW_BUTTON class object.

Example

```
#include <ui_win.hpp>

ExampleFunction1(UIW_BUTTON *button)
{



    button->DataSet("&Close");
```

UIW_BUTTON::DrawItem

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual advanced function is used to draw the object. If the WOS_OWNERDRAW status is set for the object, this function will be called when drawing the button. This allows the programmer to derive a new class from UIW_BUTTON and handle the drawing of the button, if desired.

- $returnValue_{out}$ is a response based on the success of the function call. If the object is drawn the function returns a non-zero value. If the object is not drawn, 0 is returned.

- $event_{in}$ contains the run-time message that caused the object to be redrawn. *event, region* contains the region in need of updating. The following logical events may be sent to the **DrawItem( )** function:

  S_CURRENT, S_NON_CURRENT, S_DISPLAY_ACTIVE and **S_DISPLAY_INACTIVE**—Messages that cause the object to be redrawn.

  WM_DRAWITEM—A message that causes the object to be redrawn. This message is specific to Windows and OS/2.

  Expose—A message that causes the object to be redrawn. This message is specific to Motif.

- $ccode_{in}$ contains the logical interpretation of *event.*

UIW_BUTTON::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the button object. It is declared virtual so that any derived button class can override its default operation.

- *returnValue*$_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the button object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event():**

  L_BEGIN_SELECT—Indicates that the end-user began the selection of the object by pressing the mouse button down while on the object. If the BTF_-DOWN_CLICK flag is set, the user function will be called.

  L_CONTINUE_SELECT—Indicates that the end-user previously clicked down on the object with the mouse and is now continuing to hold the mouse button down while on the object.

  L_DOUBLE_CLICK—Indicates that the object was double-clicked. If the BTF_DOUBLE_CLICK flag is set, the button's user function will be called.

L_END_SELECT—Indicates that the selection process, initiated with the L_-BEGIN_SELECT message, is complete. For example, the end-user has pressed and released the mouse button. The user function will be called.

L_SELECT—Indicates that the object has been selected. The selection may be the result of a mouse click or a keyboard action.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_CURRENT—Causes the object to draw itself to appear current. A button usually has a focus rectangle around the text or bitmap when it is current or it may have a thick border. This message is sent by the Window Manager to a window when it becomes current. The window, in turn, passes this message to the object on the window that is current.

S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager sends this message to a window when the window is subtracted from the Window Manager. The window, in turn, relays the message to all objects attached to it.

S_DISPLAY_ACTIVE—Causes the object to draw itself to appear active. An active object is one that is on the active (i.e., current) window. Most objects do not display differently whether they are active or inactive. An active object should not be confused with a current object. An object is active if it is on the active window. However, it may not be the current object on the window.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed by the event overlaps the region of the object. This message is sent by a UIW_WINDOW to all the non-current, active objects attached to it.

S_DISPLAY_INACTIVE—Causes the object to draw itself to appear inactive. An inactive object is one that is not on the active (i.e., current) window. Most objects do not display differently whether they are inactive or active.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed with the event overlaps the region of the object. This message is sent by a UIW WINDOW to all the objects attached to it.

S_HIDE_DEFAULT—Causes the object to draw as a normal button when it has been drawing as the default button. The default button has a thick border. This message is sent by another object when the object wishes to appear as the default.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_MOVE—Causes the object to update its location. The distance to move is contained in the *position* field of UI_EVENT. For example, an *event.position.-line* of -10 and an *event.position.column* of 15 moves the object 10 lines up and 15 columns to the right.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another field or window.

S_REDISPLAY—Causes the object to redraw.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

S_SHOW_DEFAULT—Causes the object to draw as the default button. The default button has a thick border. This message is sent when another button has been current and displaying as the default button but is no longer current.

S_VERIFY_STATUS—Causes the object to correlate its state (i.e., selected or not selected) with the operating system.

All other events are passed by Event( ) to UI_WINDOW_OBJECT::Event( ) for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event( )** function.

UIW_BUTTON::Information

Syntax

```
#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST request, void *data,
    ZIL_OBJECTID objectID = ID_DEFAULT);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the button:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object. Set data to ZIL_NULLP(void).

I_CLEAR_FLAGS—Clears the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be cleared, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to clear the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_-OBJECT. If the BTF_FLAGS are to be cleared, *objectID* should be ID_-BUTTON. This allows the object to process the request at the proper level. This request only clears those flags that are passed in; it does not simply clear the entire field.

I_CHANGED_STATUS—Informs the object that the programmer has changed some status flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's status flags, particularly if the new status flag settings will change the visual appearance of the object. If this request is sent, *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer changes the WOS_STATUS of an object, *objectID* should be ID_-WINDOW_OBJECT. If the BTS_STATUS is modified, *objectID* should be ID_BUTTON. This allows the object to process the request at the proper level.

I_CLEAR_STATUS—Clears the current status flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIS_STATUS that contains the status flags to be cleared, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to clear the WOS_STATUS of an object, *objectID* should be ID_-WINDOW_OBJECT. If the BTS_STATUS is to be cleared, *objectID* should be ID_BUTTON. This allows the object to process the request at the proper level. This request only clears those status flags that are passed in; it does not simply clear the entire field.

I_COPY_TEXT—Copies the *text* associated with the object into a buffer provided by the programmer. If this request is sent, *data* must be the address of a buffer where the string's text will be copied. This buffer must be large enough to contain all of the characters associated with the button and the terminating NULL character.

I_GET_BITMAP_ARRAY—Returns a pointer to the button's bitmap array. If a bitmap does not exist, NULL is returned. If this message is sent, *data* must be a pointer to **ZIL_UINT8.**

I_GET_BITMAP_HEIGHT—Returns the button's bitmap height. If this message is sent, *data* must be a pointer to a variable of type **int.**

I_GET_BITMAP_WIDTH—Returns the button's bitmap width. If this message is sent, *data* must be a pointer to a variable of type **int.**

I_GET_FLAGS—Requests the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to obtain the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the BTF_FLAGS are desired, *objectID* should be ID_BUTTON. This allows the object to process the request at the proper level.

I_GET_STATUS—Requests the current status flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIS_STATUS, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to obtain the WOS_STATUS of an object, *objectID* should be ID_WINDOW_OBJECT. If the BTS_STATUS is desired, *objectID* should be ID_BUTTON. This allows the object to process the request at the proper level.

I_GET_TEXT—Returns a pointer to the text associated with the object. If this request is sent, *data* should be a doubly-indirected pointer to ZIL_ICHAR. This request does not copy the text into a new buffer.

I_GET_VALUE—Returns the *value* associated with the button. If this message is sent, *data* must be a pointer to a variable of type EVENT_TYPE where the button's value will be copied.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_BITMAP_ARRAY—Sets the bitmap array associated with a bitmap button. If this message is sent, *data* must be a pointer to an array of ZIL_UINT8 that contains the button's new bitmap.

I_SET_BITMAP_HEIGHT—Sets the button's bitmap height. If this message is sent, *data* must be a pointer to a variable of type **int** that contains the bitmap's height.

I_SET_BITMAP_WIDTH—Sets the button's bitmap width. If this message is sent, *data* must be a pointer to a variable of type **int** that contains the bitmap's width.

I_SET_FLAGS—Sets the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be set, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to set the WOF_-FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the BTF_-FLAGS are to be set, *objectID* should be ID_BUTTON. This allows the object to process the request at the proper level. This request only sets those flags that are passed in; it does not clear any flags that are already set.

I_SET_STATUS—Sets the current status flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIS_STATUS that contains the status flags to be set, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to set the WOS_STATUS of an object, *objectID* should be ID_WINDOW_-OBJECT. If the BTS_STATUS is to be set, *objectID* should be ID_BUTTON. This allows the object to process the request at the proper level. This request only sets those status flags that are passed in; it does not clear any flags that are already set.

I_SET_TEXT—Sets the text associated with the object. This request will also redisplay the object with the new text, *data* should be a pointer to the new text.

I_SET_VALUE—Sets the *value* associated with the button. If this message is sent, *data* must be a pointer to a variable of type EVENT_TYPE that contains the button's new value.

All other requests are passed by **Information( )** to **UI_WINDOW_OBJECT::-Information( )** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information( )** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID*, the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include <ui_win.hpp>
#include <string.h>

ExampleFunction()
{
        ZIL_ICHAR string[30];
        button->Information(I_COPY_TEXT, string, ID_BUTTON);

        button1->Information(I_SET_TEXT, "&New", ID_BUTTON);
        button2->Information(I_SET_TEXT, "E&xit", ID_BUTTON);

}
```

UIW_BUTTON::Message

Syntax

#include <ui_win.hpp>

EVENT_TYPE Message(UI_WINDOW_OBJECT **object*, UI_EVENT *&event,*
EVENT_TYPE *ccode);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function serves as a user function that is called when buttons that have the BTF_-SEND_MESSAGE flag set are selected. A button whose BTF_SEND_MESSAGE flag is set will put an event whose type is the button's value on the event queue when selected. This function should not be used directly by the programmer; rather, it is used by the library.

- $returnValue_{out}$ is the *ccode* argument.

- $event_{in}$ contains the run-time message that caused the button to be selected.

- $ccode_{in}$ contains the logical interpretation of the event's type.

Example

```
ExampleFunction()
{
    // By setting the BTF_SEND_MESSAGE flag, the button's value will
    //be placed on the event queue when the button is selected.
    UIW_BUTTON = new UIW_BUTTON(1, 5, 20, "E&xit", BTF_AUTO_SIZE |
        BTF_NO_TOGGLE | BTF_SEND_MESSAGE, WOF_JUSTIFY_CENTER, NULL, L_EXIT);



}
```

UIW_BUTTON::SetDecorations

Syntax

#include <ui_win.hpp>

void SetDecorations(const ZIL_ICHAR **decorationName);*

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function sets the decorations to be used by the object. The images for the object will be loaded and the object's *myDecorations* member will be updated to point to the new ZIL_DECORATION object. By default, the object uses the images identified in the **IMG_DEF.CPP** file, which compiles into the library. (If different default images are desired, simply copy a **IMG_<ISO>.CPP** file from the OpenZinc\SOURCE\INTL directory to the \OpenZinc\SOURCE directory, and rename it to **IMG_DEF.CPP** before compiling the library.) The images are loaded from the **I18N.DAT** file, so it must be shipped with your application.

- *decorationName*$_{in}$ is the two-letter ISO country code identifying which images the object should use.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_BUTTON::UIW_BUTTON

Syntax

#include <ui_win.hpp>

UIW_BUTTON(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));

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_BUTTON by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a button is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume* |.

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UIJWIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_BUTTON::Load

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_BUTTON from a persistent object data

file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" *of Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UIJWIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{inn}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:.userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW BUTTON::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume* |. If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_BUTTON::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New()** function.

- *returnValue*$_{out}$ is a pointer to the object's **New( )** function.

UIW_BUTTON::Store

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- $name_{in}$ is the name of the object to be stored.

- $file_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

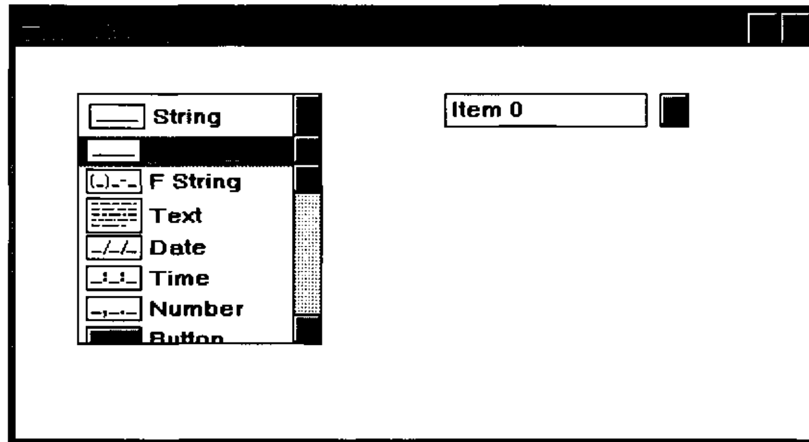
- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:.-userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 4 - UIW_COMBO_BOX

The UIW_COMBO_BOX class is used to provide a finite list of options to the end-user without taking up lots of screen space. The combo box normally only displays the current selection and a button that is used to view the list of selections. When the button is selected, a drop-down list containing all the options appears. The end-user can scroll through the list. If the end-user types some characters, the combo box will move to the option that most closely matches the characters typed. When one of the options is selected, it is placed in the default selection field and the drop-down list disappears. The options can consist of string objects or graphical objects, such as icons and bitmap buttons. The figure below shows the graphical implementation of a UIW_COMBO_BOX object:



The UIW_COMBO_BOX class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_COMBO_BOX : public UIW_WINDOW
{
public:
    static ZIL_ICHAR _className[];
    UIW_VT_LIST list;

    UIW_COMBO_BOX(int left, int top, int width, int height,
        ZIL_COMPARE_FUNCTION compareFunction =
            ZIL_NULLF(ZIL_COMPARE_FUNCTION),
        WNF_FLAGS wnFlags = WNF_NO_WRAP, WOF_FLAGS woFlags = WOF_NO_FLAGS,
        WOAF_FLAGS woAdvancedFlags = WOAF_NO_FLAGS);
    UIW_COMBO_BOX(int left, int top, int width, int height,
        ZIL_COMPARE_FUNCTTON compareFunction, WOF_FLAGS flagSetting,
        UI_ITEM *item);
    virtual ~UIW_COMBO_BOX(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual void Destroy(void);
```

```
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTTD objectID = ID_DEFAULT);
    virtual void Sort(void);

# if defined (ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFur.ction (void) ;
    static U1_WIND0W_0BJF,CT *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_TTEM),
        UI_ITEM *userTable = ZIL_NULLP(UI_1TEM));
    UIW_COMBO_BOX(const ZIL_ICHAR *name, ZIL_STORAGE_READ_ONLY *file,
        ZIL_STORAGE_OBJECT_READ_ONLY *object,
        UI_|'TEM *objectTabl e = ZIL_NULLP(UI_ITEM) ,
        UI_ITEM *userTable = ZIL_NULLP(UI_ITEM));
    virtual void Load(const ZIL_ICHAR *name, ZIL_STORAGS_READ_ONLY *file,
        ZIL_STORAGE_OBJECT_READ_ONLY *object, UI_ITEM *objectTable,
        U1_ITEM *userTable);
#endif
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTab1e,
        U1_ITEM *userTable);
#endif

    // List members.
    UI_WINDOW_0BJECT *Add(U1_WIND0W_03JECT *object);
    int Count(void);
    UI_WINDOW_OBJECT *Current(void);
    UI_WTNDOW_OBJEC? *First(void) ;
    UI_WIND0W_03JECT *Get(int index);
    int Index(UI_WINDOW_OBJECT const *element);
    UI_WINDOW_0BJECT *Last(void) ;
    UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT *object);
    UIW_COMBO_BOX &operator+(UI_WINDOW_OBJECT *object);
    UIW_COMBO_BOX &operator-(UI_WINDOW_OBJECT *object);
} ;
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_COMBO_BOX class, *_className* is "UIW_COMBO_BOX."

- *list* maintains the list of options displayed by the combo box.

UIW_COMBO_BOX::UIW_COMBO_BOX

Syntax

#include <ui_win.hpp>

UIW_COMBO_BOX(int *left,* int *top,* int *width,* int *height,*
  ZIL_COMPARE_FUNCTION *compareFunction* =
    ZIL_NULLF(ZIL_COMPARE_FUNCTION),
  WNF_FLAGS *wnFlags* = WNF_NO_WRAP,
  WOF_FLAGS *woFlags* = WOF_NO_FLAGS,
  WOAF_FLAGS *woAdvancedFlags* = WOAF_NO_FLAGS)
  or
UIW_COMBO_BOX(int *left,* int *top,* int *width,* int *height,*
  ZIL_COMPARE_FUNCTION *compareFunction,* WOF_FLAGS *flagSetting,*
  UI_ITEM **item);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These overloaded constructors create a new UIW_COMBO_BOX class object.

The first overloaded constructor creates a UIW_COMBO_BOX object.

- $left_{in}$ and top_{in} is the starting position of the combo box. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the combo box. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $height_{in}$ is the height of the combo box's drop-down list. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

compareFunction$_{in}$ is a programmer defined function that will be called by the library when sorting the list of objects attached to the combo box. *compareFunction* is called as each individual object is added and if the list is sorted explicitly by calling the **Sort( )** function. The objects can be sorted based on any key unique to the object. Pointers to the objects being compared are passed to the *compareFunction,* so any information required to do the sorting needs to be associated with the object. Because the objects can be of any type, even a derived type, the object pointers will need to be typecast in the *compareFunction.*

The definition of the *compareFunction* is as follows:

> int FunctionName(void **element1,* void **element2);*
>
> *returnValue*$_{out}$ indicates the relative ordering of the two elements. *returnValue* should be negative if *element1* should be placed in front of *element2,* 0 if the two elements are sorted the same or positive if *element1* should come after *element2.*
>
> *element1*$_{in}$ is a pointer to the first element to be compared. This void pointer must be typecast according to the type of object being sorted.
>
> *element2*$_{in}$ is a pointer to the second element to be compared. This void pointer must be typecast according to the type of object being sorted.

wnFlags$_{in}$ are flags that define the operation of the combo box drop-down list. The following flags (declared in **UI_WIN.HPP)** affect the operation of a UIW_COMBO_-BOX class object's drop-down list:

> **WNF_AUTO_SELECT—**Causes each object in the list to be automatically selected when it becomes current, thus placing it in the default selection field of the combo box. This flag is always set on the combo box, whether it is set by the programmer or not.
>
> **WNF_AUTO_SORT—**Causes the combo box options to be sorted in alphabetical order.
>
> **WNF_BITMAP_CHILDREN—**Indicates that some of the objects contain bitmaps. Setting this flag will affect the spacing of objects in the drop-down list. Normally, objects are spaced according to a pre-determined cell height value. If this flag is set, however, the objects will be spaced according to the actual height of the objects.

WNF_CONTINUE_SELECT—Allows the end-user to drag through the drop-down list options with the mouse button pressed. If this flag is not set, the highlight on the list options will not follow the dragging mouse. This flag should usually be set on a combo box.

WNF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WNF_-FLAGS.

WNF_NO_WRAP—Will not allow arrowing up or down to wrap from the end of the list to the beginning or vice versa.

- *woFlags*$_{in}$ are flags (common to all window objects) that determine the general operation of the combo box object. The following flags (declared in UI_WIN.HPP) control the general presentation of, and interaction with, a UIW_COMBO_BOX class object:

WOF_AUTO_CLEAR—Automatically marks the entire buffer if the end-user tabs to the field from another object. If the user then enters data (without first having pressed any movement or editing keys) the entire field will be replaced.

WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is 1/10th the size of a normal cell by default.

WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_SELECTABLE—Prevents the object from being selected. Typically, the object will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

WOF_VIEW_ONLY—Prevents the object from being edited. However, the object may become current and the user may scroll through the data, mark it, and copy it.

- *woAdvancedFlags*$_{in}$ are flags (general to all window objects) that determine the advanced operation of the combo box object.

WOAF_NO_FLAGS—Does not associate any special advanced flags with the window object. This flag should not be used in conjunction with any other WOAF flags.

WOAF_NON_CURRENT—Prevents the object from becoming current. If this flag is set, users will not be able to select the combo box from the keyboard. The combo box may still be selected using the mouse, but it will not become current.

The second overloaded constructor creates a combo box using a pre-defined item array. These items are used to create UIW_STRING objects.

- $left_{in}$ and top_{in} is the starting position of the combo box. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the combo box. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $height_{in}$ is the height of the combo box's drop-down list. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $compareFunction_{in}$ is a programmer defined function that will be called by the library when sorting the list of objects attached to the combo box. For more details, see the description of *compareFunction* with the first constructor.

- $flagSetting_{in}$ is a value that is checked against each UI_ITEM's *value* field. If the item's *value* field is the same as *flagSetting,* that item is marked as the selected option in the combo box.

- $item_{in}$ is an array of UI_ITEM structures that are used to construct a set of string items within the combo box. For more information regarding the use of the UI_ITEM structure, see "Chapter 18—UI_ITEM" in *Programmer's Reference Volume 1.*

Example

```
#include <ui_win.hpp>

void ExampleFunction (UI_WINDOW_MANAGER *windowManager)
{
    UIW_WTNDOW *window = U1W_WIND0W::Generic(0, C, 45, 8, "Window");
```

```
    *window
        + new UIW_PROMPT(2, 1, "Combo box:")
        + &(* new UIW_COMBO_BOX(20, 1, 17, 6)
            + new UIW_SCROLL_BAR(0, 0, 0, 0, SBF_VERTICAL)
            + new UIW_STRING(0, 0, 15, "Combo 1")
            + new UIW_STRING(0, 0, 15, "Combo 2")
            + new UIW_STRING(0, 0, 15, "Combo 3")
            + new UIW_STRING(0, 0, 15, "Combo 4")
            + new UIW_STRING(0, 0, 15, "Combo 5")
            + new UIW_STRING(0, 0, 15, "Combo 6")
            + new UIW_STRING(0, 0, 15, "Combo 7")
            + new UIW_STRING(0, 0, 15, "Combo 8")
            + new UIW_STRING(0, 0, 15, "Combo 9")
            + new UIW_STRING(0, 0, 15, "Combo 10"));
    *windowManager + window;
}
```

UIW_COMBO_BOX::~UIW_COMBO_BOX

Syntax

#include <ui_win.hpp>

virtual ~UIW_COMBO_BOX(void);

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_COMBO_-BOX object. All objects attached to the combo box will also be destroyed.

UIW_COMBO_BOX::Add
UIW_COMBO_BOX::operator +

Syntax

#include <ui_win.hpp>

UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);*

or

UIW_COMBO_BOX &operator + (UI_WINDOW_OBJECT **object);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These overloaded functions are used to add an object to the combo box. If a compare function was specified for the combo box, it will be used to insert the object in the proper location. If no compare function was specified, however, the object will be added to the end of the list.

The first function adds an object to the UIW_COMBO_BOX.

- $returnValue_{out}$ is a pointer to *object* if the addition was successful. Otherwise, *returnValue* is NULL.

- $object_{in}$ is a pointer to the object to be added to the combo box.

The second overloaded operator adds an object to the UIW_COMBO_BOX. This operator overload is equivalent to calling the UIW_COMBO_BOX::Add() function except that it allows the chaining of object additions to the UIW_COMBO_BOX.

- $returnValue_{out}$ is a pointer to the UIW_COMBO_BOX. This pointer is returned so that the operator may be used in a statement containing other operations.

- $object_{in}$ is a pointer to the object that is to be added to the combo box.

UIW_COMBO_BOX::ClassName

Syntax

#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

> *retumValue*$_{out}$ is a pointer to *_className.*

UIW_COMBO BOX::Count

Syntax

```
#include <ui_win.hpp>

int Count(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function returns a count of the number of objects in the combo box list.

> *returnValue*$_{out}$ is the number of objects in the list.

UIW_COMBO_BOX::Current

Syntax

#include <ui_win.hpp>

UI_WINDOW_OBJECT *Current(void);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function returns a pointer to the current object, if one exists, in the combo box.

- $returnValue_{out}$ is a pointer to the current object in the combo box. If there is no current object, *returnValue* is NULL.

UIW_COMBO_BOX::Destroy

Syntax

#include <ui_win.hpp>

virtual void Destroy (void);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function destroys all the objects attached to the combo box.

UIW_COMBO_BOX::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the combo box object. It is declared virtual so that any derived combo box class can override its default operation.

- *returnValue*$_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the combo box object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by Event( ):

    L_BEGIN_SELECT—Indicates that the end-user began the selection of the object by pressing the mouse button down while on the object.

    L_CANCEL—Causes the drop-down list to be closed.

    L_DOWN—Makes the next object in the list the current selection. This message is interpreted from a keyboard event.

L_FIRST—Causes the first object in the combo box list to be made current.

L_LAST—Causes the last object in the combo box list to be made current.

L_NEXT—The combo box object processes this message by suppressing it. This allows the combo box's parent window to process it. This message is interpreted from a keyboard event.

L_PGDN—Causes the list to scroll down a page and make the selected object the new bottom item on the page. This message is interpreted from a keyboard event.

L_PGUP—Causes the list to scroll up a page and make the selected object the new top item on the page. This message is interpreted from a keyboard event.

L_PREVIOUS—The combo box object processes this message by suppressing it. This allows the combo box's parent window to process it. This message is interpreted from a keyboard event.

L_SELECT—Causes the drop-down list to appear. This message is interpreted from a keyboard event.

L_UP—Makes the next object in the list the current selection. This message is interpreted from a keyboard event.

S_ADD_OBJECT—Causes a new object to be added to the combo box. *event.data* will point to the new object to be added.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_CURRENT—Causes the object to draw itself to appear current. This message is sent by the Window Manager to a window when it becomes current. The window, in turn, passes this message to the object on the window that is current.

S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager sends this message to a window when the window is subtracted from the Window Manager. The window, in turn, relays the message to all objects attached to it.

S_DISPLAY_ACTIVE—Causes the object to draw itself to appear active. An active object is one that is on the active (i.e., current) window. Most objects do not display differently whether they are active or inactive. An active object should not be confused with a current object. An object is active if it is on the active window. However, it may not be the current object on the window.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed by the event overlaps the region of the object.

S_DISPLAY_INACTIVE—Causes the object to draw itself to appear inactive. An inactive object is one that is not on the active (i.e., current) window. Most objects do not display differently whether they are inactive or active.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed with the event overlaps the region of the object.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another field or window.

S_REDISPLAY—Causes the object to redraw.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

S_RESET_DISPLAY—Changes the display to a different resolution, *event.data* should point to the new display class to be used. If *event.data* is NULL, then a text mode display will be created. This event is specific to DOS and must be placed on the event queue by the programmer. The library will never generate this event.

S_SUBTRACT_OBJECT—Causes an object to be subtracted from the combo box. *event.data* will point to the object to be subtracted.

All other events are passed by Event( ) to UIW_WINDOW::Event() for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived Event() function.

UIW_COMBO_BOX::First

Syntax

```
#include <ui_win.hpp>

UI_WINDOW_OBJECT  *First(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function returns a pointer to the first object, if one exists, in the combo box.

- $returnValue_{out}$ is a pointer to the first object in the combo box. If there is no first object, *returnValue* is NULL.

UIW_COMBO_BOX::Get

Syntax

#include <ui_win.hpp>

UI_WINDOW_OBJECT *Get(int *index);*

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This functions us used to get a pointer to a specific object in the combo box.

- *returnValue*$_{out}$ is a pointer to the object whose index value is *index.* If no object is at the index position specified by *index*, NULL is returned.

- *index*$_{in}$ is the index value of the object to be located. The list is zero-based, so the first object in the list has an index value of 0.

UIW_COMBO_BOX::Index

Syntax

#include <ui_gen.hpp>

int Index(UI_WINDOW_OBJECT const **element);*

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function returns the index value of the specified object. If no object matches the specified object, -1 is returned.

- *returnValue*$_{out}$ gives the index of the object in the UIW_COMBO_BOX object. List element indexes are zero based (i.e., the first element in a list has an index value of 0). If *element* is not found in the UIW_COMBO_BOX object, -1 is returned.

- *element*$_{in}$ is a pointer to the object to find.

UIW_COMBO_BOX::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request,* void **data,*
ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in UI_WIN.HPP) are recognized by the window:

I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

I_COPY_TEXT—Copies the *text* associated with the current selection into a buffer provided by the programmer. If this request is sent, *data* must be the address of a buffer where the selection's text will be copied. This buffer must be large enough to contain all of the characters associated with the object and the terminating NULL character.

I_DESTROY_LIST—Destroys all non-support objects attached to the combo box. This request simply calls Destroy().

I_GET_BITMAP_ARRAY—Returns a pointer to the current selection's bitmap array. If a bitmap does not exist, NULL is returned. If this message is sent, *data* must be a pointer to ZIL_UINT8.

I_GET_CURRENT—Returns a pointer to the current object in the combo-box.

I_GET_FIRST—Returns a pointer to the first object in the combo-box.

I_GET_LAST—Returns a pointer to the last object in the combo-box.

I_GET_NUMBERID_OBJECT—Returns a pointer to an object whose *numberID* matches the value in *data,* if one exists. If no object attached to the combo box has a *numberID* that matches *data,* NULL is returned. If this message is sent, *data* must be a pointer to a NUMBERID.

I_GET_STRINGID_OBJECT—Returns a pointer to an object whose *stringID* matches the character string in *data,* if one exists. If no object attached to the combo box has a *stringID* that matches *data*, NULL is returned. If this message is sent, *data* must be a pointer to a string.

I_GET_SUPPORT_CURRENT—Returns a pointer to the current support object in the combo-box.

I_GET_SUPPORT_FIRST—Returns a pointer to the first support object in the combo-box.

I_GET_SUPPORT_LAST—Returns a pointer to the last support object in the combo-box.

I_GET_TEXT—Returns a pointer to the text associated with the current selection. If this request is sent, *data* should be a doubly-indirected pointer to ZIL_ICHAR. If *data* is NULL, the selection's text pointer will be returned as *returnValue.* This request does not copy the text into a new buffer.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_RESET_SELECTION—Causes the combo box to update its current selection field to match the current object in the list.

I_SET_BITMAP_ARRAY—Sets the bitmap array associated with the current selection. If this message is sent, *data* must be a pointer to an array of ZIL_UINT8 that contains the new bitmap.

I_SET_TEXT—Sets the text associated with the current selection. This request will also redisplay the object with the new text, *data* should be a pointer to the new text.

All other requests are passed by Information() to UIW_WINDOW::Information( ) for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the Information() function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include <ui_win.hpp>
ExampleFunction()
{
        UIW_COMBO_BOX *comboBox;
```

```
		WNF_FLAGS wnFlags;
		comboBox->Information(I_GET_FLAGS, &wnFlags, ID_WINDOW);


	}
```

UIW_COMBO_BOX::Last

Syntax

#include <ui_win.hpp>

UI_WINDOW_OBJECT *Last(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function returns a pointer to the last object, if one exists, in the combo box.

- *returnValue*$_{out}$ is a pointer to the last object in the combo box. If there is no last object, *returnValue* is NULL.

UIW_COMBO_BOX::Sort

Syntax

#include <ui_gen.hpp>

void Sort(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function sorts the UIW_COMBO_BOX object using the *compareFunction* that was assigned in the constructor. If the list has no compare function, no sort occurs.

UIW_COMBO_BOX::Subtract
UIW_COMBO_BOX::operator -

Syntax

```
#include <ui_win.hpp>

UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT *object);
    or

UIW_COMBO_BOX &operator - (UI_WINDOW_OBJECT *object);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These overloaded functions are used to subtract an object from the combo box. These functions do not delete the objects; they merely remove them from the list. The programmer is responsible for destroying any objects explicitly subtracted from the combo box.

The first function subtracts an object from the UIW_COMBO_BOX.

- *returnValue*$_{out}$ is a pointer to *object* if the subtraction was successful. Otherwise, *returnValue* is NULL.

- *object*$_{in}$ is a pointer to the object to be subtracted from the combo box.

The second overloaded operator subtracts an object from the UIW_COMBO_BOX. This operator overload is equivalent to calling the **UIW_COMBO_BOX::Subtract(**) function except that it allows the chaining of object subtractions from the UIW_COMBO_BOX.

- *returnValue*$_{out}$ is a pointer to the UIW_COMBO_BOX. This pointer is returned so that the operator may be used in a statement containing other operations.

- *object*$_{in}$ is a pointer to the object that is to be subtracted from the combo box.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_COMBO_BOX::UIW_COMBO_BOX

Syntax

#include <ui_win.hpp>

UIW_COMBO_BOX(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));

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This advanced constructor creates a new UIW_COMBO_BOX by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a combo box is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *Ul_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_COMBO_BOX::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_COMBO_BOX from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_COMBO_BOX::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_COMBO_BOX::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New()** function.

- *returnValue*$_{out}$ is a pointer to the object's **New()** function.

UIW COMBO_BOX::Store

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" *of Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

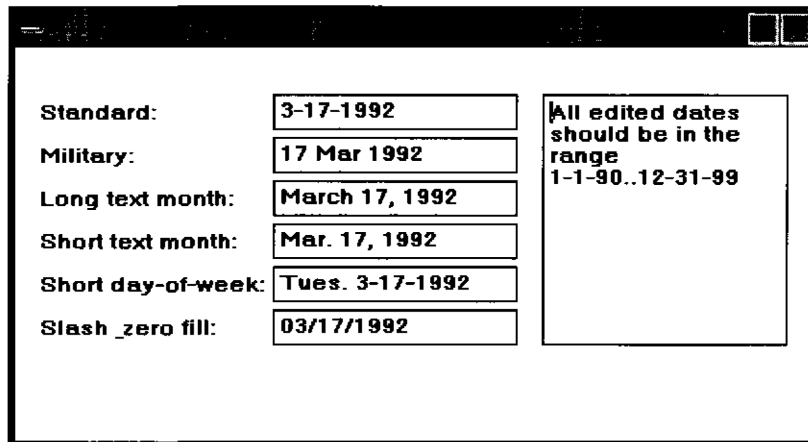
- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the

description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 5 - UIW_DATE

The UIW_DATE class is used to display date information to the screen and to collect information, in date form, from an end-user. The UIW_DATE class will automatically format the displayed date. The UIW_DATE class is a high-level object that is used to interact with the end-user. It makes use of the ZIL_DATE class, which is a low-level object that handles the details of date data manipulation. See "Chapter 51—ZIL_DATE" of *Programmer's Reference Volume 1* for more information about the ZIL_DATE class. The figure below shows the graphical implementation of a window with several variations of the UIW_DATE class object:



The UIW_DATE class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_DATE : public UIW_STRING
{
public:
    static ZIL_ICHAR _className[];
    static int defaultlnitialized;
    DTF_FLAGS dtFlags;
#if defined(ZIL_3x_COMPAT)
    static DTF_FLAGS rangeFlags;
#endif

    UIW_DATE(int left, int top, int width, ZIL_DATE *date,
        const ZIL_ICHAR *range = ZIL_NULLP(ZIL_ICHAR),
        DTF_FLAGS dtFlags = DTF_NO_FLAGS,
        WOF_FLAGS woFlags = WOF_BORDER | WOF_AUTO_CLEAR,
        ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION));
    virtual ~UIW_DATE(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    ZIL_DATE *DataGet(void);
    void DataSet(ZIL_DATE *date);
```

```
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);
    virtual int Validate(int processError = TRUE);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_DATE(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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

    void SetLanguage(const ZIL_ICHAR *languageName);
protected:
    ZIL_DATE *date;
    ZIL_ICHAR *range;
    const ZIL_LANGUAGE *myLanguage;
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_DATE class, *_className* is "UIW_DATE."

- *defaultInitialized* indicates if the default language strings for this object have been set up. The default strings are located in the file LANG_DEF.CPP. If *defaultInitialized* is TRUE, the strings have been set up. Otherwise they have not been.

- *rangeFlags* are flags that define how the range values are interpreted. *rangeFlags* is set to DTF_US_FORMAT by default.

- *dtFlags* are flags that define the operation of the UIW_DATE class. A full description of the date flags is given in the UIW_DATE constructor.

- *date* is a pointer to a ZIL_DATE that is used to manage the low-level date information. If the WOF_NO_ALLOCATE_DATA flag is set, this member will simply point to the ZIL_DATE value passed in.

- *range* is a string that specifies the range(s) of acceptable date values, *range* is a copy of the range that is passed to the constructor.

- *myLanguage* is the ZIL_LANGUAGE object that contains the string translations for this object.

UIW_DATE::UIW_DATE

Syntax

```
#include <ui_win.hpp>

UIW_DATE(int left, int top, int width, ZIL_DATE *date,
    const ZIL_ICHAR *range = ZIL_NULLP(ZIL_ICHAR),
    DTF_FLAGS dtFlags = DTF_NO_FLAGS,
    WOF_FLAGS woFlags = W0FJ30RDER | WOF_AUTO_CLEAR,
    ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_DATE class object.

- $left_{in}$ and top_{in} is the starting position of the date field within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the date field. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value. The height of the object is determined automatically by the UIW JDATE object.

- *date*$_{in}$ is a pointer to a ZIL_DATE object. Its value will be used as the initial value.

- *range*$_{in}$ is a string that specifies the valid date ranges. A range consists of a minimum value, a maximum value, and the values in between. The date must be specified with the format YYYY-MM-DD, where YYYY is the four-digit year, MM is the month and DD is the day of the month. For example, if a range of "1990-1-1.. 1990-12-31" is specified, the UIW_DATE class object will only accept those date values that fall between January 1, 1990 and December 31, 1990, inclusive. Open-ended ranges can be specified by leaving the minimum or maximum value off. For example, a range of "1990-1-1.." will allow all dates that are January 1, 1990 or thereafter. Multiple, disjoint ranges can be specified by separating the individual ranges with a slash (i.e., '/'). For example, "1990-1-1..1990-1-31/ 1990-3-1.." will accept all dates during January 1990 and dates of March 1, 1990 and later. If *range* is NULL, any date within the absolute range (100-1-1 through 32767-12-31) is accepted. This string is copied by the UIW_DATE class object to the *range* member variable.

- *dtFlags*$_{in}$ describes how the date should display and interpret the date information. The following flags (declared in UI_GEN.HPP) control the general presentation of a UIW_DATE class object:

DTF_ALPHA_MONTH—Causes the month name to be spelled-out, as opposed to being represented numerically.

```
March 28, 1990
December 4, 1980
January 3, 2003
```

DTF_DASH—Separates the date fields with a dash, regardless of the default country date separator.

```
3-28-1990
12-04-1980
1-3-2003
```

DTF_DAY_OF_WEEK—Causes a spelled-out day-of-week to be shown in the date.

```
Monday May 4, 1992
Friday Dec. 5, 1980
Su&day Jan. 4, 2003
```

DTF_EUROPEAN_FORMAT—Forces the date to be formatted in the European format (i.e., *day/month/year*), regardless of the default country information.

```
28/3/1990
4 December, 1980
3 Jan., 2003
```

DTF_ASIAN_FORMAT—Forces the date to be formatted in the Asian format (i.e., *year/-month/day),* regardless of the default country information.

```
1990/3/28
1980 December 4
2003 Jan. 3
```

DTF_MILITARY_FORMAT—Forces the date to be formatted in the United States Air Force format, regardless of the default country information. The air force format is ordered by *day month year* where *month* is either a 3-letter abbreviated word and *year* is a two-digit year value (if the DTF_SHORT_YEAR or DTF_-SHORT_MONTH flags are set) or *month* is spelled-out and *year* is a four-digit value. The air force style is used as the default. However, in order to accommodate the formats used in other branches of the military, other date formatting options (e.g., zero fill, upper case, etc.) may be used in conjunction with the standard military format.	`(air force style-default)`
	`4 Jul 91`
	`4 July 1991`

DTF_NO_FLAGS—Does not associate any special flags with the date object. In this case, the date will be formatted using the default country information. This flag should not be used in conjunction with any other DTF flags.	`(European format)`
	`4 December 1989`
	`23 June 2000`
	
	`(Asian format)`
	`1989 December 4`
	`2000 June 23`

DTF_SHORT_DAY—Adds an abbreviated day-of-week to the date.	`Wed. March 28, 1990`
	`Thurs. Dec. 4, 1980`
	`Sat. January 3, 2003`

DTF_SHORT_MONTH—Adds an abbreviated month name to the date.	`Mar. 28, 1990`
	`Dec. 4, 1980`
	`Jan. 3, 2003`

DTF_SHORT_YEAR—Forces the year to be formatted as a two-digit value.	`3/28/90`
	`December 4,`
	`Jan. 3, 89`

DTF_SLASH—Separates the date fields with a slash, regardless of the default country date separator.	`3/28/90`
	`12/04/1900`
	`1/3/2003`

DTF_SYSTEM—Uses the system date.	`3/28/90`
	`12/04/1980`
	`1/3/2003`

DTF_UPPER_CASE—Converts the alphabetic date characters to upper-case.	`MARCH 28, 1990`
	`DEC. 4, 1980`
	`SATURDAY JAN 3, 2003`

DTF_US_FORMAT—Forces the date to be formatted in the U.S. format (i.e., *month/day/year),* regardless of the default country information.

```
March 28, 1990
12/4/1980
Jan 3, 2003
```

DTF_ZERO_FILL—Forces the year, month and day values to be zero filled when their values are less than 10.

```
March 08, 1990
12/04/1980
01/03/2003
```

- *woFlags*$_{in}$ are flags (common to all window objects) that determine the general operation of the date object. The following flags (declared in UI_WIN.HPP) affect the operation of the UIW_DATE class object:

  WOF_AUTO_CLEAR—Automatically marks the entire buffer if the end-user tabs to the field from another object. If the user then enters data (without first having pressed any movement or editing keys) the entire field will be replaced. This flag is set by default in the constructor.

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOF_INVALID—Sets the initial status of the field to be "invalid." Invalid entries fit in the absolute range determined by the object type but do not fulfill all the requirements specified by the program. For example, a date may initially be set to 1-1-80, but the final date, edited by the end-user, must be in the range "1-1-90..12-31-99." The initial date in this example fits the absolute range requirements of a UIW_DATE class object but does not fit into the specified range. By denoting the field as invalid, you force the user to enter an acceptable value.

  WOF_JUSTIFY_CENTER—Center-justifies the data within the displayed field.

  WOF_JUSTIFY_RIGHT—Right-justifies the data within the displayed field.

  WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying

an object's position in minicell coordinates. A minicell is l/10th the size of a normal cell by default.

WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

WOF_NO_FLAGS—Does not associate any special window flags with the object. Setting this flag left-justifies the data. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor edit the date information. Typically, the field will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

WOF_SUPPORT_OBJECT—Causes the object to be placed in the parent object's support list. The support list is reserved for objects that are not displayed as part of the user region of the window. Care should be used when setting this flag on an object that does not use it by default as undesirable effects may occur. This flag generally should not be used by the programmer.

WOF_UNANSWERED—Sets the initial status of the field to be "unanswered." An unanswered field is displayed as an empty field.

WOF_VIEW_ONLY—Prevents the object from being edited. However, the object may become current and the user may scroll through the data, mark it, and copy it.

- *userFunction*$_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. The user function will be called by the library when:

  1—the user moves onto the field,

  2—the <ENTER> key is pressed while the field is current or, if the field is in a list, the mouse is clicked on it, or

3—the user moves to a different field in the window or to a different window.

Because the user function is called at these times, the programmer can do data validation or any other type of necessary operation. The definition of the *user-Function* is as follows:

EVENT_TYPE FunctionName(UI_WINDOW_OBJECT **object,*
UI_EVENT *&event,* EVENT_TYPE *ccode);*

$returnValue_{out}$ indicates if an error has occurred. *returnValue* should be 0 if the no error occurred. Otherwise, the programmer should call the error system with an appropriate error message and return -1.

$object_{in}$ is a pointer to the object for which the user function is being called. This argument must be typecast by the programmer if class-specific members need to be accessed.

$event_{in}$ is the run-time message passed to the object.

$ccode_{in}$ is the logical or system code that caused the user function to be called. This code (declared in **UI_EVT.HPP)** will be one of the following constant values:

L_SELECT—The <ENTER> key was pressed while the field was current or, if the field is in a list, the mouse was clicked on the field.

S_CURRENT—The object just received focus because the user moved to it from another field or window. This code is sent before any editing operations are permitted.

S_NON_CURRENT—The object just lost focus because the user moved to another field or window.

NOTE: If a user function is associated with the object, **Validate()** must be called explicitly from within *userFunction* if range checking is desired.

Example

```
#include <ui_win.hpp>

ExampleFunction1(UI_WINDOW_MANAGER *windowManager)
{
    ZIL_DATE date; // system date
    ZIL_TIME time; // system time
```

```
    // Create a window with a date and time field.
    UIW_WINDOW *window = UIW_WINDOW::Generic(0, 0, 45, 8, NULL, "Window");
    *window
        + new UIW_PROMPT(2, 1, "Date..")
        + new UIW_DATE(9, 1, 20, &date, ZIL_NULLP(ZIL_ICHAR),
            DTF_AIiPHA_MONTH | DTF_SYSTEM)
        + new UIW_PROMPT(2, 3, "Time..")
        + new UIW_TIME(9, 3, 20, &time, ZIL_NULLP(ZIL_ICHAR), TMF_SECONDS);

    // The date object will automatically be destroyed when the window
    //is destroyed.
}
```

UIW_DATE::~UIW_DATE

Syntax

#include <ui_gen.hpp>

virtual ~UIW_DATE(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_DATE object.

UIW_DATE::ClassName

Syntax

#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to *_className.*

UIW_DATE::DataGet

Syntax

#include <ui_win.hpp>

ZIL_DATE *DataGet(void);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • Motif • Curses • NEXTSTEP

Remarks

This function gets the current date information associated with the UIW_DATE class object.

- *returnValue*$_{out}$ is a pointer to a ZIL_DATE object containing the current date value.

Example

```
ttinclude <ui_win.hpp>

ExampleFunction(UIW_DATE  *dateObject)
```

```
    ZIL_DATE *date = dateObject->DataGet();

}
```

UIW_DATE::DataSet

Syntax

#include <ui_win.hpp>

void DataSet(const ZIL_DATE **date);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- Motif
- Curses
- NEXTSTEP

Remarks

This function assigns a new value to the UIW_DATE object and redisplays the field. If no value is passed in (i.e., *value* is NULL), the field will be redrawn.

- $value_{in}$ is a pointer to the new date. If the WOF_NO_ALLOCATE_DATA flag is set, this argument must be a ZIL_DATE, allocated by the programmer, that is not destroyed until the UIW_DATE class object is destroyed. Otherwise, the information associated with this argument is copied by the UIW_DATE class object. If this argument is NULL, no date information is changed, but the date field is redisplayed.

Example

```
#include <ui_win.hpp>
ExampleFunctionl(UIW_DATE *date)
{


    ZIL_DATE datelnfo(92, 10, 19);
    date->DataSet(fcdatelnfo);
}
```

UIW_DATE::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the date object. It is declared virtual so that any derived date class can override its default operation.

- *returnValue*$_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the date object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event():**

  L_SELECT—Indicates that the object has been selected. The selection may be the result of a mouse click or a keyboard action.

  S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

  S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another field or window.

  All other events are passed by **Event()** to **UIW_STRING::Event()** for processing.

UIW_DATE::Information

Syntax

```
#include <ui_win.hpp>

void *Information(ZIL_INFO_REQUEST request, void *data,
    ZIL_OBJECTID objectID = ID_DEFAULT);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WFN.HPP)** are recognized by the date object:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_CLEAR_FLAGS—Clears the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be cleared, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to clear the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_-OBJECT. If the STF_FLAGS are to be cleared, *objectID* should be ID_-STRING. This allows the object to process the request at the proper level. This

request only clears those flags that are passed in; it does not simply clear the entire field.

I_DECREMENT_VALUE—Decrements the date's value. If this message is sent, *data* must be a pointer to an integer. The date object's value will be decremented by the value of *data.* The date will not be modified if the new value is not within the specified range.

I_GET_FLAGS—Requests the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to obtain the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the STF_FLAGS are desired, *objectID* should be ID_STRING. This allows the object to process the request at the proper level.

I_GET_VALUE—Gets the current value for the object. If this request is sent, *data* should be a pointer to ZIL_DATE. If *data* is NULL *returnValue* will return a pointer to ZIL_DATE.

I_INCREMENT_VALUE—Increments the date's value. If this message is sent, *data* must be a pointer to an integer. The date object's value will be incremented by the value of *data.* The date will not be modified if the new value is not within the specified range.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_FLAGS—Sets the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be set, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to set the WOF_-FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the STF_-FLAGS are to be set, *objectID* should be ID_STRING. This allows the object to process the request at the proper level. This request only sets those flags that are passed in; it does not clear any flags that are already set.

I_SET_VALUE—Sets the current value for the object. If this request is sent, *data* should be a pointer to a ZIL_DATE that contains the value to be set.

All other requests are passed by **Information()** to **UIW_STRING::Information()** for processing.

- *$data_{ir/out}$* is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- *$objectID_{in}$* is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include <ui_win.hpp>

ExampleFunction()
{
        UIW_DATE *date, *datel, *date2;



        ZIL_ICHAR string[30];
        date-information(I_COPY_TEXT, &string);



        date1->Informat ion(I_SET_TEXT, "12/31/93");
        date2->Information(I_SET_TEXT, "10/19/93")j



}
```

UIW_DATE::SetLanguage

Syntax

#include <ui_win.hpp>

void SetLanguage(const ZIL_ICHAR *languageName);*

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function sets the language to be used by the object. The string translations for the object will be loaded and the object's *myLanguage* member will be updated to point to the new ZIL_LANGUAGE object. By default, the object uses the language identified in the **LANG_DEF.CPP** file, which compiles into the library. (If a different default language is desired, simply copy a **LANG_<ISO>.CPP** file from the OpenZinc\SOURCE\-INTL directory to the \OpenZinc\SOURCE directory, and rename it to **LANG_DEF.CPP** before compiling the library.) The language translations are loaded from the **I18N.DAT** file, so it must be shipped with your application.

- *languageName*$_{in}$ is the two-letter ISO language code identifying which language the object should use.

UIW_PATE::Validate

Syntax

```
#include <ui_win.hpp>

virtual int Validate(int processError = TRUE);
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• Motif	• Curses	• NEXTSTEP

Remarks

This function is used to validate objects. When an object receives the S_CURRENT or S_NON_CURRENT messages, it calls **Validate()** to check if the value entered is valid.

However, if a user function is associated with the object, **Validate()** must be called explicitly from the user function if range checking is desired. The value is invalid if it is not within the absolute range of the object or if it is not within a range specified by the *range* member variable.

- $returnValue_{out}$ indicates the result of the validation. The possible values for *returnValue* are:

  DTI_AMBIGUOUS—The month name was ambiguous (e.g., "01-JU-92").

  DTI_GREATER_THAN_RANGE—The date was greater than the maximum value of a negatively open-ended range.

  DTI_INVALID—An invalid date format was encountered (e.g., "31 Jan, 1992").

  DTI_INVALID_NAME—Either the month name or the day-of-week name was invalid (e.g., "Tuesday Jaan 28, 1992" or "Tyesday Jan 28, 1992").

  DTI_LESS_THAN_RANGE—The date was less than the minimum value of a positively open-ended range.

  DTI_OK—The date was entered in a correct format and within the valid range.

  DTI_OUT_OF_RANGE—The date value was out of range (e.g., "Jan 33, 1992").

  DTI_VALUE_MISSING—The required date value was missing (e.g., "5, 1991").

- $processError_{in}$ determines whether **Validate()** should call UI_ERROR_SYSTEM::-ReportError() if an error occurs. If *processError* is TRUE, ReportError() is called. Otherwise, the error system is not called.

Example

```
#include <ui_win.hpp>

EVENT_TYPE DateUserFunction(UI_WINDOW_OBJECT *object, UI_EVENT &,
    EVENT_TYPE ccode)
{
    if (ccode != S_NON_CURRENT)
        return (ccode);

    / / o specific validation.
    ZIL_DATE currentDate;
```

```
ZIL_DATE *date = ((UIW_DATE *)object)->DataGet();

// Call the default Validate function to check for valid date,
int valid = object->Validate(TRUE);

// Call error system if the date entered is later than the system date.
if (valid == DTI_OK && currentDate < *date)
{
    valid = DTI_INVALID;
    ZIL_ICHAR dateString[64];
    currentDate.Export(dateString, 64, DTF_NO_FLAGS);
    object->errorSystem->ReportError(object->windowManager, WOS_NO_STATUS,
        "The date must be before %s.", dateString);
}

// Return error status,
if (valid == DTI_OK)
    return (0);
else
    return (-1);
```

Storage Members

This section describes those class members that are used for storage purposes.

UIW_DATE::UIW_DATE

Syntax

#include <ui_win.hpp>

UIW_DATE(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));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_DATE by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a date is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_DATE::Load

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_DATE from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_DATE::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_DATE::Store

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

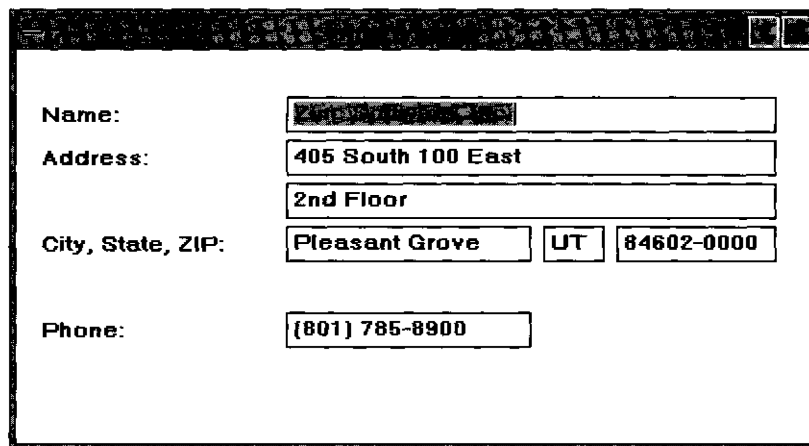
- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT: .objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 6 - UIW_FORMATTED_STRING

The UIW_FORMATTED_STRING class is used to display and collect formatted string information. Formatting is done on a keystroke-by-keystroke basis. Only those characters allowed by the format can be typed. The cursor position automatically jumps past non-editable formatting characters (e.g., parentheses in the area code of a phone number). The figure below shows the graphical implementation of two UIW_FORMATTED_STRING class objects:



The UIW_FORMATTED_STRING class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_FORMATTED_STRING : public UIW_STRING
{
public:
    static ZIL_ICHAR _className[];

    UIW_FORMATTED_STRING(int left, int top, int width,
        ZIL_ICHAR *compressedText, ZIL_ICHAR *editMask,
        ZIL_ICHAR *deleteText,
        WOF_FLAGS woFlags = WOF_BORDER | WOF_AUTO_CLEAR,
        ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION));
    virtual ~UIW_FORMATTED_STRING(void);
    virtual ZIL_ICHAR *ClassName(void);
    ZIL_ICHAR *DataGet(int compressedText = FALSE);
    void DataSet(ZIL_ICHAR *text);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    void Export(ZIL_ICHAR *destination, int expanded);
    FMI_RESULT Import(ZIL_ICHAR *source);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_FORMATTED_STRING(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
#if defined (ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

protected:
    ZIL_ICHAR *compressedText;
    ZIL_ICHAR *editMask;
    ZIL_ICHAR *deleteText;
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_FORMATTED_STRING class, *_className* is "UIW_FORMATTED_-STRING."

- *compressedText* is the raw, unformatted text string.

- *editMask* defines the format for the string. Each character in *editMask* denotes the type of character that can appear in that location. The character can either be editable or it can be a literal (e.g., the parentheses in the area code of a phone number).

- *deleteText* contains the characters that will appear if no text has been entered in the string (i.e., *compressedText* is empty). This text serves as placeholder text if no data has been entered or if characters are deleted.

UIW_FORMATTED_STRING::UIW_FORMATTED_STRING

Syntax

#include <ui_win.hpp>

UIW_FORMATTED_STRING(int *left,* int *top,* int *width,* ZIL_ICHAR **compressedText,*
ZIL_ICHAR **editMask,* ZIL_ICHAR **deleteText,*
WOF_FLAGS *woFlags* = WOF_BORDER | WOF_AUTO_CLEAR,
ZIL_USER_FUNCTION *userFunction* = ZIL_NULLF(ZIL_USER_FUNCTION));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_FORMATTED_STRING class object.

- $left_{in}$ and top_{in} is the starting position of the formatted string field within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the formatted string. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value. The height of the string is determined automatically by the UIW_FORMATTED_STRING object.

- $compressedText_{in}$ is an initial raw text string that will be formatted according to the *editMask* and *deleteText* arguments. This string is copied into a buffer allocated by the UIW_FORMATTED_STRING object unless the WOF_NO_ALLOCATE_DATA flag is set, in which case the buffer containing *compressedText* is used.

- $editMask_{in}$ defines the format for the string. Each character in *editMask* denotes the type of character that can appear in that location. The character can either be editable or it can be a literal (e**.g.,** the parentheses in the area code of a phone number). This string is always copied by the UIW_FORMATTED_STRING class object. Valid characters used to define the edit mask are:

a—Allows the end-user to enter a space (' ') or any letter (i.e., 'a' through 'z' or 'A' through 'Z').

A—Same as the 'a' character option except that a lower-case letter is automatically converted to an upper-case letter.

c—Allows the end-user to enter a space (' '), a number (i.e., '0' through '9') or any alphabetic character (i.e., 'a' through 'z' or 'A' through 'Z').

C—Same as the 'c' character option except that a lower-case character is automatically converted to upper-case.

L—Uses this position as a literal place holder. Using this character causes the formatted string to get the character to be read and displayed from the *deleteText.* The end-user cannot position on nor edit this character.

N—Allows the end-user to enter any numeric digit.

x—Allows the end-user to enter any printable character.

X—Same as the 'x' character option except that a lower-case letter is automatically converted to an upper-case alphanumeric character.

$deleteText_{in}$ contains the characters that will appear if no text has been entered in the string (i.e., *compressedText* is empty). This text serves as placeholder text if no data has been entered or if characters are deleted. It also specifies the characters to be used as literal, uneditable characters.

$woFlags_{in}$ are flags (common to all window objects) that determine the general operation of the formatted string object. The following flags (declared in **UI_WIN.HPP)** control the general presentation of, and interaction with, a UIW_-FORMATTED_STRING class object:

WOF_AUTO_CLEAR—Automatically marks the entire buffer if the end-user tabs to the field from another object. If the user then enters data (without first having pressed any movement or editing keys) the entire field will be replaced. This flag is set by default in the constructor.

WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support

Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

WOF_INVALID—Sets the initial status of the field to be "invalid." Invalid entries fit in the absolute range determined by the object type but do not fulfill all the requirements specified by the program. By default, all formatted string information is valid. For example, a formatted string field for a phone number may initially be set to (...) ...-...., but the final string edited by the end-user must contain a valid phone number. In this case the initial string information does not fulfill the program's requirements.

WOF_JUSTIFY_CENTER—Center-justifies the data within the displayed field.

WOF_JUSTIFY_RIGHT—Right-justifies the data within the displayed field.

WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is l/10th the size of a normal cell by default.

WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

WOF_NO_FLAGS—Does not associate any special window flags with the object. Setting this flag left-justifies the data. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor edit the formatted string information. Typically, the field will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

WOF_SUPPORT_OBJECT—Causes the object to be placed in the parent object's support list. The support list is reserved for objects that are not displayed as part of the user region of the window. Care should be used when setting this flag on an object that does not use it by default as undesirable effects may occur. This flag generally should not be used by the programmer.

WOF_UNANSWERED—Sets the initial status of the field to be "unanswered." An unanswered field is displayed as an empty field.

WOF_VIEW_ONLY—Prevents the object from being edited. However, the object may become current and the user may scroll through the data, mark it, and copy it.

- *userFunction*$_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. The user function will be called by the library when:

  1—the user moves onto the field,

  2—the <ENTER> key is pressed while the field is current or, if the field is in a list, the mouse is clicked on it, or

  3—the user moves to a different field in the window or to a different window.

  Because the user function is called at these times, the programmer can do data validation or any other type of necessary operation. The definition of the *userFunction* is as follows:

```
EVENT_TYPE FunctionName(UI_WINDOW_OBJECT *object,
    UI_EVENT &event, EVENT_TYPE ccode);
```

  returnValue$_{out}$ indicates if an error has occurred. *returnValue* should be 0 if the no error occurred. Otherwise, the programmer should call the error system with an appropriate error message and return -1.

  object$_{in}$ is a pointer to the object for which the user function is being called. This argument must be typecast by the programmer if class-specific members need to be accessed.

  event$_{in}$ is the run-time message passed to the object.

  ccode$_{in}$ is the logical or system code that caused the user function to be called. This code (declared in **UI_EVT.HPP)** will be one of the following constant values:

  L_SELECT—The <ENTER> key was pressed while the field was current, or, if the field is in a list, the mouse was clicked on the field.

S_CURRENT—The object just received focus because the user moved to it from another field or window. This code is sent before any editing operations are permitted.

S_NON_CURRENT—The object just lost focus because the user moved to another field or window.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Add formatted string fields to a window.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_BORDER
        + new UIW_TITLE(" Sample strings ")
        + new UIW_PROMPT(2, 1, "Formatted strings..")
        + new UIW_FORMATTED_STRING(22, 1, 41, "8017858900",     // phone number
            "LNNNLLNNNLNNNN", "(...)...-____")
        + new urw_FORMATTED_STRING(43, 2, 20, "840620000",      // zip code
            "NNNNNLNNNN", ".....-____");
    *windowManager + window;

    // The formatted strings will automatically be destroyed when the window
    // is destroyed.
}
```

UIW_FORMATTED_STRING::~UIW_FORMATTED_STRING

Syntax

#include <ui_win.hpp>

virtual ~UIW_FORMATTED_STRING(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_FORMATTED_STRING object.

UIW_FORMATTED_STRING::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- $returnValue_{out}$ is a pointer to *_className.*

UIW_FORMATTED_STRING::DataGet

Syntax

```
#include <ui_win.hpp>

ZIL_ICHAR *DataGet(int compressedText = FALSE);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function gets the current text information associated with the UIW_FORMATTED_-STRING class object.

- *returnValue*$_{out}$ is a pointer to the formatted string buffer.

- *compressedText*$_{in}$ indicates if the returned text is formatted or not. If *compressedText* is TRUE, no formatting will be performed on the returned text. Otherwise the returned text will be formatted according to the *editMask* and the *deleteText.*

Example

```
#include <ui_win.hpp>

static int CheckAreaCode(UI_WINDOW_OBJECT *data, UI_EVENT &event,
    EVENT_TYPE ccode)
{
    // Only look for a non-current message.
    if (ccode != S_NON_CURRENT)
        return (FMI_OK);     // Continue with program.

    // Make sure the area code is not 000.
    UIW_FORMATTED_STRING *StringField = (UIW_FORMATTED_STRING *)data;
    ZIL_ICHAR *number = stringField->DataGet(TRUE);
    if (number[0] != '0' II number[1] != '0' II number[2] != '0')
        return (FMI_OK);
    data->errorSystem->ReportError(stringField->windowManager, WOF_UNANSWERED,
        "The phone number you entered, %s, does not have a valid area code.",
        stringField->DataGet(FALSE));
    return (FMI_INVALID);


ExampleFunctionl(UI_WINDOW_MANAGER *windowManager)
{
    // Create a phone number string field.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 8);
    *window
        + new UIW_BORDER
        + new UIW_TITLE(" Sample strings ")
        + new UIW_PROMPT(2, 1, "Phone number:")
        + new UIW_FORMATTED_STRING(16, 1, 15, "8017858900",
            "LNNNLLNNNLNNNN", "(...) ...-...."), WOF_NO_FLAGS, CheckAreaCode);
    *windowManager + window;
```

UIW_FORMATTED_STRING::DataSet

Syntax

#include <ui_win.hpp>

void DataSet(ZIL_ICHAR **text);*

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function assigns new text to the UIW_FORMATTED_STRING object and redisplays the field. If no value is passed in (i.e., *text* is NULL), the field will be redrawn.

- $text_{in}$ is a pointer to the new text information for the formatted string. This string must conform to the *editMask* and *deleteText* specified for the formatted string. If the WOF_NO_ALLOCATE_DATA flag is set, this argument must be space, allocated by the programmer, that is not destroyed until the UIW_FORMATTED_STRING class object is destroyed. Otherwise, the information associated with this argument is copied by the UIW_FORMATTED_STRING class object. If this argument is NULL, no string information is changed, but the formatted string field is redisplayed.

Example

```
#include <ui_win.hpp>

struct COMPANY_INFO
{
    ZIL_ICHAR name[64];
    ZIL_ICHAR address1[64];
    ZIL_ICHAR address2[64] ;
    ZIL_ICHAR representative[64];
    ZIL_ICHAR phone[16];
};

ExampleFunction1(UI_WINDOW_MANAGER *windowManager)
{
    // Manually add a formatted string field to a window.
    UIW_STRING *name, *address1, *address2;
    UIW_FORMATTED_STRING *phoneNumber;
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 50, 10);
```

```
*window
    + new UIW_BORDER
    + new UIW_TITLE(" Company Information ")
    + new UIW_PROMPT(2, 1, *name:")
    +  (name = new UIW_STRING(11, 1, 30, NULL, 64))
    + new UIW_PROMPT(2, 2, "Address:")
    +  (address1 = new UIW_STRING(11, 2, 30, NULL, 64))
    +  (address2 = new UIW_STRING(11, 3, 30, NULL, 64))
    + new UIW_PROMPT(2, 5, "Representative:")
    +  (representative = new UIW_STRING(18, 5, 30, NULL, 64))
    + new UIW_PROMPT(2, 6, "Phone Number:")
    +  (phone = new UIW_FORMATTED_STRING(18, 6, 15, NULL, "LNNNLLNNNLNNNN",
        "(...)  ...-     "));


// Get the company information and set the window information.
COMPANY_INFO company;


name->DataSet(company.name);
address1->DataSet(company.address1);
address2->DataSet(company.address2);
representative->DataSet(company.representative);
phone->DataSet(company.phone);
*windowManager + window;
```

UIW_FORMATTED_STRING::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the formatted string object. It is declared virtual so that any derived formatted string class can override its default operation.

- *returnValue*$_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the formatted string object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event():**

  E_KEY—Indicates that a key has been pressed. Places the character in the formatted string, if it matches the *editMask* specifications for the current cursor position. This message is interpreted from a keyboard event.

  L_BACKSPACE—Causes the first editable character to the left of the cursor position to be deleted and moves the cursor to that position. This message is interpreted from a keyboard event.

  L_BOL—Causes the cursor to be moved to the first editable character in the string. This message is interpreted from a keyboard event.

  L_CUT—Causes the highlighted portion of the string to be cut and placed in the paste buffer. The cut text will include any literal formatting characters. This message is interpreted from a keyboard event.

  L_DELETE—Causes the marked characters, if any, or the character at the current cursor position to be deleted. If there is no character at the current cursor position (i.e., there is a character from the *deleteText* string) the first editable character to the right of the current cursor position will be deleted. The cursor's position will not change. This message is interpreted from a keyboard event.

  L_DELETE_EOL—Causes all editable characters from the current cursor position to the end of the field to be deleted. This message is interpreted from a keyboard event.

  L_END_MARK—Indicates that the end-user has finished marking text in the string. This message is interpreted from a mouse event.

  L_EOL—Causes the cursor to be moved to the last editable character in the string. This message is interpreted from a keyboard event.

  L_LEFT—Causes the cursor to be moved to the next editable character to the left of the current position. This message is interpreted from a keyboard event.

L_PASTE—Causes the contents of the paste buffer to be placed in the field at the current cursor position. The new text will be formatted to fit the *editMask* and *deleteText* specifications. This message is interpreted from a keyboard event.

L_RIGHT—Causes the cursor to be moved to the next editable character to the right of the current position. This message is interpreted from a keyboard event.

L_SELECT—Indicates that the object has been selected. The selection may be the result of a mouse click or a keyboard action.

L_WORD_LEFT—Causes the cursor position to be moved to the beginning of the current word or, if the cursor is at the beginning of the current word, to the beginning of the next word to the left of the current cursor position. This message is interpreted from a keyboard event.

L_WORD_RIGHT—Causes the cursor position to be moved to the beginning of the next word to the right of the current cursor position. This message is interpreted from a keyboard event.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_CURRENT—Causes the object to draw itself to appear current. This message is sent by the Window Manager to a window when it becomes current. The window, in turn, passes this message to the object on the window that is current.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another field or window.

All other events are passed by **Event()** to **UIW_STRING::Event()** for processing.

UIW FORMATTED STRING:;Export

Syntax

#include <ui_win.hpp>

void Export(ZIL_ICHAR **destination*, int *expanded);*

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function returns the current text from the formatted string. The text, returned by *destination,* may either be in expanded or raw form.

- *destination*$_{in}$ is a character array, allocated by the programmer, that will receive the formatted string text, *destination* must be long enough to contain the string including the NULL terminator.

- *expanded*$_{in}$ indicates whether the exported text should be expanded or not. If *expanded* is TRUE, the text will be formatted. Otherwise it will be the raw, unformatted text.

Example

```
#include <ui_win.hpp>

static int CheckAreaCode(UI_WINDOW_OBJECT *data, UI_EVENT &event,
    EVENT_TYPE ccode)
{
    // Only look for a non-current message.
    if (ccode != S_NON_CURRENT)
        return (FMI_OK);     // Continue with program.

    // Make sure the area code is not 000.
    UIW_FORMATTED_STRING *stringField = (UIW_FORMATTED_STRING *)data;
    ZIL_ICHAR number[16];
    stringField->Export(number, FALSE);
    if (number[0] != '0' II number[1] != '0' II number[2] != '0')
        return (FMI_OK);
    data->errorSystem->ReportError(stringField->windowManager, WOF_NO_FLAGS,
        "The phone number you entered, %s, does not have a valid prefix",
        stringField->DataGet(FALSE));
    return (FMI_INVALID);


ExampleFunction1(UI_WINDOW_MANAGER *windowManager)
{
    // Create a phone number string field.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 8);
    *window
        + new UIW_BORDER
        + new UIW_TITLE(" Sample strings ")
        + new UIW_PROMPT(2, 1, "Phone number:")
        + new UIW_FORMATTED_STRING(16, 1, 15, "8017858900",
            "LNNNLLNNNLNNNN", "(...) ...-...."), WOF_NO_FLAGS, CheckAreaCode);
    *windowManager + window;
```

}

UIW_FORMATTED_STRING::Import

Syntax

#include <ui_win.hpp>

FMI_RESULT Import(ZIL_ICHAR **source);*

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function sets the string information associated with the UIW_FORMATTED_-STRING class object.

- *returnValue*$_{out}$ is the result of the import operation. *returnValue* can have one of the following values:

  FMI_OK—The import operation was successful.

  FMI_INVALID_CHARACTERS—*source* contained invalid characters.

- *source*$_{in}$ is a pointer to the new string. This string must conform to the *editMask* and *deleteText* specifications. If the WOF_NO_ALLOCATE_DATA flag is set, this argument must be space, allocated by the programmer, that is not destroyed until the UIW_FORMATTED_STRING class object is destroyed. Otherwise, the information associated with this argument is copied by the UIW_FORMATTED_STRING class object.

Example

```
#include <ui_win.hpp>

struct COMPANY_INFO
{
    ZIL_ICHAR name[64];
    ZIL_ICHAR address1[64];
    ZIL_ICHAR address2[64] ;
    ZIL_ICHAR representative[64];
    ZIL_ICHAR phone[16];
};

ExampleFunction1(UI_WINDOW_MANAGER *windowManager)
{
    // Manually add a formatted string field to a window.
    UIW_STRING *name, *address1, *address2;
    UIW_FORMATTED_STRING *phoneNumber;
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 50, 10);
    *window
        + new UIW_BORDER
        + new UIW_TITLE(" Company Information ")
        + new UIW_PROMPT(2, 1, *name:")
        + (name = new UIW_STRING(11, 1, 30, NULL, 64))
        + new UIW_PROMPT(2, 2, "Address:")
        + (address1 = new UIW_STRING(11, 2, 30, NULL, 64))
        + (address2 = new UIW_STRING(11, 3, 30, NULL, 64))
        + new UIW_PROMPT(2, 5, "Representative:")
        + (representative = new UIW_STRING(18, 5, 30, NULL, 64))
        + new UIW_PROMPT(2, 6, "Phone Number:")
        + (phone = new UIW_FORMATTED_STRING(18, 6, 15, NULL, "LNNNLLNNNLNNNN",
            "(...) ...     "));

    // Get the company information and set the window information.
    COMPANY_INFO company;

    name->Import(company.name);
    address1->Import(company.address1);
    address2->Import(company,address2);
    representative->Import(company.representative);
    phone->Import(company.phone);
    *windowManager + window;
```

UIW_FORMATTED_STRING::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request*, void **data,*
ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- $returnValue_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- $request_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the formatted string object:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

  I_SET_TEXT—Sets the text associated with the object. This request will also redisplay the object with the new text, *data* should be a pointer to the new text.

  All other requests are passed by **Information( )** to **UIW_STRING::Information()** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information( )** function is virtual, it is possible for an

object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include <ui_win.hpp>
#include <string.h>

ExampleFunction()
{
        UIW_FORMATTED_STRING *fString, *fStringl, *fString2;



        ZIL_ICHAR string[30];
        fString->Information(I_COPY_TEXT, string);



        fStringl->Informat ion(I_SET_TEXT, "8017858900");
        fString2->Informat ion(I_SET_TEXT, "8017858998");



}
```

Storage Members

This section describes those class members that are used for storage purposes.

UIW_FORMATTED_STRING::UIW_FORMATTED_STRING

Syntax

#include <ui_win.hpp>

UIW_FORMATTED_STRING(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));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_FORMATTED_STRING by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a formatted string is stored in a data file it is usually stored as part of a UIW_-WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" *of Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:.userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_FORMATTED_STRING::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_FORMATTED_STRING from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::. objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_FORMATTED_STRING::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OB,JECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_FORMATTED_STRING::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's New() function.

- *returnValue*$_{out}$ is a pointer to the object's New() function.

UIW_FORMATTED_STRING::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

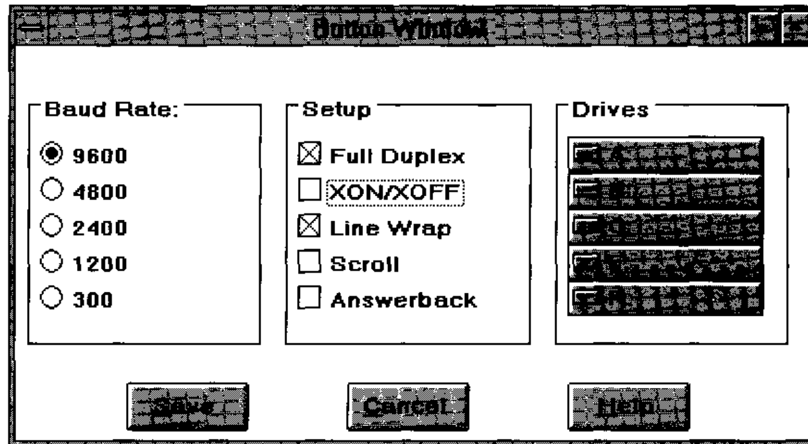
- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see

the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 7 - UIW_GROUP

The UIW_GROUP class is used to associate related objects with each other, both visually and logically. For example, a group is typically used to contain radio buttons. Visually, the end-user can see the available options and which one is selected. Logically, the end-user is able to select only one option from the group. The group has a title that can be used to further clarify the relationship of the objects in the group. To visually relate the objects, the group draws a border (or different colored background in some text modes) around the entire group. The picture below shows a graphical implementation of UIW_-GROUP objects:



The UIW_GROUP class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_GROUP : public UIW_WINDOW
{
public:
    static ZIL_ICHAR _className[];
    UIW_GROUP(int left, int top, int width, int height, ZIL_ICHAR *text,
        WNF_FLAGS wnFlags = WNF_AUTO_SELECT,
        WOF_FLAGS woFlags = WOF_NO_FLAGS);
    virtual ~UIW_GROUP(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    ZIL_ICHAR *DataGet(void);
    void DataSet(ZIL_ICHAR *text);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

#if defined(ZIL_MOTIF)
    Widget labelWidget;
#endif
#if defined (ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
```

```
    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));
    UIW_GROUP(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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif
protected:
    ZIL_ICHAR *text;

#if defined(ZIL_MOTIF)
    virtual void RegionMax(UI_WINDOW_OBJECT *object);
#endif
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_GROUP class, *_className* is "UIW_GROUP."

- *labelWidget* is a Motif widget used to display the group's title in Motif. This member is available in Motif only.

- *text* is the text that is to appear on the group's title.

UIW_GROUP::UIW_GROUP

Syntax

#include <ui_win.hpp>

UIW_GROUP(int *left*, int *top,* int *width,* int *height,* ZIL_ICHAR **text,*
    WNF_FLAGS *wnFlags* = WNF_AUTO_SELECT,
    WOF_FLAGS *woFlags*=WOF_NO_FLAGS);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This constructor creates a new UIW_GROUP object.

- $left_{in}$ and top_{in} is the starting position of the group. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the group. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $height_{in}$ is the height of the group. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $text_{in}$ is the text to display on the group's title. A hotkey for the group may be specified by inserting the '&' character into the string before the desired hotkey character. For example, if the string "Options" is to be displayed and 'O' is to be the hotkey, the string should be entered as "&Options." The '&' will not be displayed, but will cause the hotkey character to be drawn appropriately. If an '&' is required in the text that is displayed, place two '&' characters in the string (e.g., "A && B" will display as "A & B" and the group will not have a hotkey). Selecting the group using its hotkey will make the group current. This string is copied by the UIW_GROUP class unless the WOF_NO_ALLOCATE_DATA flag is set. If this flag is set, *text* must be space, allocated by the programmer, that is not deleted until the UIW_GROUP object has been deleted.

- $wnFlags_{in}$ are flags that define the operation of the group. The following flags (declared in **UI_WIN.HPP)** affect the operation of a UIW_GROUP class object:

  WNF_AUTO_SELECT—Causes each object in the group to be automatically selected when it becomes current. Typically this flag is used when the group contains radio buttons. If the end-user arrows through the radio buttons, the

current button will always be the selected button. This flag is set by default in the constructor.

WNF_AUTO_SORT—Causes the group options to be sorted in alphabetical order.

WNF_NO_FLAGS—Does not associate any special flags with the group. This flag should not be used in conjunction with any other WNF_ flags.

WNF_SELECT_MULTIPLE—Allows more than one option in the group to become selected at the same time. This flag is typically used if the group contains check boxes.

- $woFlags_{in}$ are flags (common to all window objects) that determine the general operation of the group object. The following flags (declared in **UI_WIN.HPP)** control the general presentation of, and interaction with, a UIW_GROUP class object:

WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is 1/10th the size of a normal cell by default.

WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags. This flag is set by default in the constructor.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor select any objects in the group. Typically, the object will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Create a window and add it to the Window Manager.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 62, 10);
    *window
        + new UIW_BORDER
        + new UIW_TITLE("Company Information")
        + &(*new UIW_GROUP(2, 9, 20, 6, "Group")
            + new UIW_BUTTON(1, 1, 16, "Radio-button 1", BTF_RADIO_BUTTON,
                WOF_NO_FLAGS)
            + new UIW_BUTTON(1, 2, 16, "Radio-button 2", BTF_RADIO_BUTTON,
                WOF_NO_FLAGS)
            + new UIW_BUTTON(1, 3, 16, "Radio-button 3", BTF_RADIO_BUTTON,
                WOF_NO_FLAGS)
            + new UIW_BUTTON(1, 4, 16, "Radio-button 4", BTF_RADIO_BUTTON,
                WOF_NO_FLAGS)
            + new UIW_BUTTON(1, 5, 16, "Radio-button 5", BTF_RADIO_BUTTON,
                WOF_NO_FLAGS))

    *windowManager + window;



}
```

UIW_GROUP::~UIW_GROUP

Syntax

#include <ui_win.hpp>

virtual ~UIW_GROUP(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_GROUP object. All objects attached to the group will also be destroyed.

UIW_GROUP::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This virtual function returns the object's class name.

- $returnValue_{out}$ is a pointer to *_className.*

UIW_GROUP::DataGet

Syntax

```
#include <ui_win.hpp>

ZIL_ICHAR *DataGet(void);
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function gets the text associated with the group object.

- *returnValue*$_{out}$ is a pointer to the text associated with the group.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_GROUP *groupObject)
{
    ZIL_ICHAR *text = groupObject->DataGet();


}
```

UIW_GROUP::DataSet

Syntax

#include <ui_win.hpp>

void DataSet(ZIL_ICHAR **text);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function assigns new text to the group and redisplays the group. If no text is passed in (i.e., *text* is NULL), the group will be redrawn.

- *text*$_{in}$ is a pointer to the new text information to be displayed on the group. If the WOF_NO_ALLOCATE_DATA flag is set, *text* must be a string, allocated by the programmer, that is not destroyed until the UIW_GROUP class object is destroyed. Otherwise, the information associated with this argument is copied by the UIW_-GROUP class object.

Example

```
#include <ui_win.hpp>

ExampleFunction1(UIW_GROUP *group)
{
    ZIL_ICHAR text[] = "Baud Rates";
    group->DataSet(text);
}
```

UIW_GROUP::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the group object. It is declared virtual so that any derived group class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the group object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by Event( ):

  L_DOWN—Moves the focus down one object. This message is interpreted from a keyboard event.

L_LEFT—Moves the focus left one object. This message is interpreted from a keyboard event.

L_NEXT—The group object processes this message by suppressing it. This allows the group's parent window to process it. This message is interpreted from a keyboard event.

L_PREVIOUS—The group object processes this message by suppressing it. This allows the group's parent window to process it. This message is interpreted from a keyboard event.

L_RIGHT—Moves the focus right one object. This message is interpreted from a keyboard event.

L_UP—Moves the focus up one object. This message is interpreted from a keyboard event.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_CURRENT—Causes the object to draw itself to appear current. This message is sent by the Window Manager to the window when it becomes current. The window, in turn, passes this message to the object on the window that is current. If the group receives this message it sends it to the current object on the group.

S_DISPLAY_ACTIVE—Causes the object to draw itself to appear active. An active object is one that is on the active (i.e., current) window. Most objects do not display differently whether they are active or inactive. An active object should not be confused with a current object. An object is active if it is on the active window. However, it may not be the current object on the window.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed by the event overlaps the region of the object.

This message is sent by the window to all the non-current, active objects attached to it.

S_DISPLAY_INACTIVE—Causes the object to draw itself to appear inactive. An inactive object is one that is not on the active (i.e., current) window. Most objects do not display differently whether they are inactive or active.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed with the event overlaps the region of the object. This message is sent by the window to all the objects attached to it.

S_MOVE—Causes the object to update its location. The distance to move is contained in the *position* field of UI_EVENT. For example, an *event.position.-line* of -10 and an *event.position.column* of 15 moves the object 10 lines up and 15 columns to the right.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another window.

S_REDISPLAY—Causes the object to redraw.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

All other events are passed by **Event()** to **UIW_WINDOW::Event()** for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event()** function.

UIW_GROUP::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request,* void **data,*
ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in UI_WIN.HPP) are recognized by the group:

    I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

    I_COPY_TEXT—Copies the *text* associated with the group's title into a buffer provided by the programmer. If this request is sent, *data* must be the address of a buffer where the group's text will be copied. This buffer must be large enough to contain all of the characters associated with the group and the terminating NULL character.

    I_GET_TEXT—Returns a pointer to the text associated with the group's title. If this request is sent, *data* should be a doubly-indirected pointer to ZIL_ICHAR. If *data* is NULL, the group's tex *returnValue*. This request does not copy the text into a new buffer.

    I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_TEXT—Sets the text associated with the group's title. This request will also redisplay the object with the new text, *data* should be a pointer to the new text.

All other requests are passed by **Information**() to **UIW_WINDOW::Information()** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID*, the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include <ui_win.hpp>
#include <string.h>

ExampleFunction()
{
        UIW_GROUP *groupl, *group2;
        char string[30];

        groupl->Information(I_COPY_TEXT, string);
        group2-information(I_SET_TEXT, "Select Baud Rate:");



}
```

UIW_GROUP::RegionMax

Syntax

#include <ui_win.hpp>

virtual void RegionMax(UI_WINDOW_OBJECT **object);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function calculates how much space *object* can occupy within the group and sets *object.trueRegion* accordingly. The regions occupied by objects that have the WOF_NON_-FIELD_REGION flag set are not included in the calculation since their regions are reserved. The regions of any other objects, however, are still available and included in the total region, since these objects can overlap with others.

- *object*$_{in}$ is a pointer to the object that is requesting the maximum region of the group. Its trueRegion member will be modified with its actual position.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_GROUP::UIW_GROUP

Syntax

```
#include <ui_win.hpp>

UIW_GROUP(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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_GROUP by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a group is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_GROUP::Load

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_GROUP from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UIJVINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_G ROUP::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_GROUP::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New()** function.

- *returnValue*$_{out}$ is a pointer to the object's **New()** function.

UIW_GROUP::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

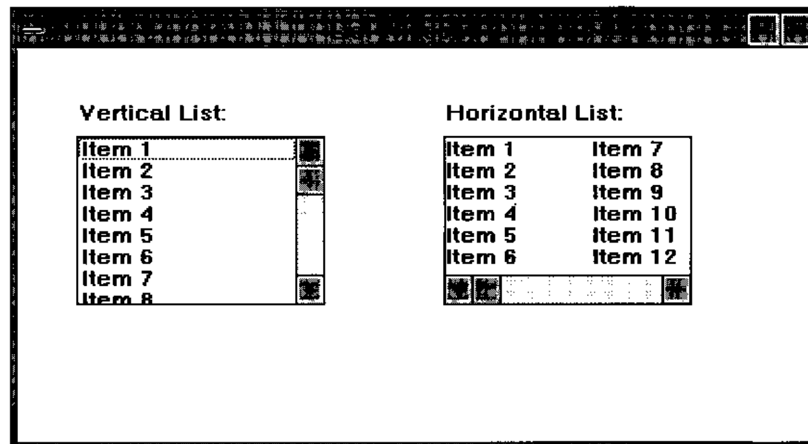
- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see

the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *$userTable_{in}$* is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 8 - UIW HZ LIST

The UIW_HZ_LIST class is a selection object used to present a list of objects to the end-user. The objects can be text only or may contain a bitmap or icon. The horizontal list will position the objects in multiple vertical columns, filling each column from top-to-bottom before placing any objects in the next column. A horizontal scroll bar can be added to the list to allow scrolling with the mouse. A typical use for the horizontal list is to present a list of items, perhaps file names, and to allow the end-user to select one or more of the items. The figure below shows the graphical implementation of a UIW_-HZJLIST object with several string objects:



The UIW_HZ_LIST class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_HZ_LIST : public UIW_WINDOW
{
public:
    static ZIL_ICHAR _className[];

    UIW_HZ_LIST(int left, int top, int width, int height,
        int cellWidth, int cellHeight,
        ZIL_COMPARE_FUNCTION compareFunction =
            ZIL_NULLF(ZIL_COMPARE_FUNCTION),
        WNF_FLAGS wnFlags = WNF_NO_WRAP | WNF_CONTINUE_SELECT,
        WOF_FLAGS woFlags = WOF_BORDER,
        WOAF_FLAGS woAdvancedFlags = WOAF_NO_FLAGS);
    UIW_HZ_LIST(int left, int top, int width, int height,
        ZIL_COMPARE_FUNCTION compareFunction, WOF_FLAGS flagSetting,
        UI_ITEM *item);
    virtual ~UIW_HZ_LIST(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual void Destroy(void);
    virtual EVENT_TYPE Event (const UI_EVENT Seventh-
    virtual void information(ZIL_INFO_REQUEST request, void *data,
```

```
        ZIL_OBJECTID objectID = ID_DEFAULT);
    virtual void Sort(void);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_HZ_LIST(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
#if defined (ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

    // List members.
#if defined (ZIL_MACINTOSH)
    UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);
#endif

protected:
    int cellWidth;
    int cellHeight;

#if defined (ZIL_MSDOS) || defined (ZIL_CURSES)
public:
    virtual EVENT_TYPE ScrollEvent(UI_EVENT &event);
#endif
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_HZ_LIST class, *_className* is "UIW_HZ_LIST."

- *cellWidth* is the width of each cell or column in the horizontal list. If the horizontal list is wider than the cell width, the list will have multiple columns.

- *cellHeight* is the height of each cell or row in the horizontal list. If the horizontal list is taller than the cell height, the list will have multiple rows.

UIW_HZ_LIST::UIW_HZ_LIST

Syntax

```
#include <ui_win.hpp>

UIW_HZ_LIST(int left, int top, int width, int height, int cellWidth, int cellHeight,
    ZIL_COMPARE_FUNCTION compareFunction =
        ZIL_NULLF(ZIL_COMPARE_FUNCTION),
    WNF_FLAGS wnFlags = WNF_NO_WRAP,
    WOF_FLAGS woFlags = WOF_BORDER,
    WOAF_FLAGS woAdvancedFlags - WOAF_NO_FLAGS);
    or
UIW_HZ_LIST(int left, int top, int width, int height,
    ZIL_COMPARE_FUNCTION compareFunction, WOF_FLAGS flagSetting,
    UI_ITEM *item);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These overloaded constructors create a new UIW_HZ_LIST object.

The <u>first</u> overloaded constructor creates a UIW_HZ_LIST object.

- $lejt_{in}$ and top_{in} is the starting position of the horizontal list. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_m$ is the width of the horizontal list. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $height_{in}$ is the height of the horizontal list. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- *cellWidth*$_{in}$ is the width of each cell or column in the horizontal list. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- *cellHeight*$_{in}$ is the height of each cell or row in the horizontal list. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- *compareFunction*$_{in}$ is a programmer defined function that will be called by the library when sorting the list of objects attached to the horizontal list. *compareFunction* is called as each individual object is added and if the list is sorted explicitly by calling the **Sort( )** function. The objects can be sorted based on any key unique to the object. Pointers to the objects being compared are passed to the *compareFunction,* so any information required to do the sorting needs to be associated with the object. Because the objects can be of any type, even a derived type, the object pointers will need to be typecast in the *compareFunction.*

  The definition of the *compareFunction* is as follows:

  int FunctionName(void **element1,* void **element2);*

  returnValue$_{out}$ indicates the relative ordering of the two elements. *returnValue* should be negative if *element1* should be placed in front of *element2,* 0 if the two elements are sorted the same or positive if *element1* should come after *element2.*

  element1$_{in}$ is a pointer to the first element to be compared. This void pointer must be typecast according to the type of object being sorted.

  element2$_{in}$ is a pointer to the second element to be compared. This void pointer must be typecast according to the type of object being sorted.

- *wnFlags*$_{in}$ are flags that define the operation of the horizontal list. The following flags (declared in **UI_WIN.HPP)** affect the operation of a UIW_HZ_LIST class object:

  WNF_AUTO_SELECT—Causes each object in the list to be automatically selected when it becomes current. This flag is typically used when radio buttons are added to the horizontal list.

  WNF_AUTO_SORT—Causes the horizontal list options to be sorted in alphabetical order.

WNF_BITMAP_CHILDREN—Indicates that some of the objects contain bitmaps. Setting this flag will affect the spacing of objects in the list. Normally, objects are spaced according to a pre-determined cell height value. If this flag is set, however, the objects will be spaced according to the actual height of the objects. This flag should be set when adding check boxes or radio buttons to the horizontal list.

WNF_CONTINUE_SELECT—Allows the end-user to drag through the list options with the mouse button pressed. If this flag is not set, the highlight on the list options will not follow the dragging mouse.

WNF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WNF_-FLAGS.

WNF_NO_WRAP—Will not allow arrowing up, down, left or right to wrap from the end of the list to the beginning or vice versa.

WNF_SELECT_MULTIPLE—Allows more than one object to be selected at a time.

- *woFlags*$_{in}$ are flags (common to all window objects) that determine the general operation of the horizontal list object. The following flags (declared in **UI_WIN.HPP)** control the general presentation of, and interaction with, a UIW_HZ_-LIST class object:

WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles.

WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is 1/10th the size of a normal cell by default.

WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

WOF_NON_SELECTABLE—Prevents the object from being selected. Typically, the object will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

- *woAdvancedFlags*$_{in}$ are flags (general to all window objects) that determine the advanced operation of the horizontal list object.

WOAF_NO_FLAGS—Does not associate any special advanced flags with the window object. This flag should not be used in conjunction with any other WOAF flags.

WOAF_NON_CURRENT—Prevents the object from becoming current. If this flag is set, users will not be able to select the horizontal list from the keyboard. The horizontal list may still be selected using the mouse, but it will not become current.

The second overloaded constructor creates a horizontal list using a pre-defined item array. These items are used to create UIW_STRING objects. The horizontal list's *cellWidth* is set to the list's width and *cellHeight* is set to 1.

- *left*$_{in}$ and *top*$_{in}$ is the starting position of the horizontal list. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- *width*$_{in}$ is the width of the horizontal list. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- *height*$_{in}$ is the height of the horizontal list. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- *compareFunction*$_{in}$ is a programmer defined function that will be called by the library when sorting the list of objects attached to the horizontal list. For more details, see the description of *compareFunction* with the first constructor.

- *flagSetting*$_{in}$ is a value that is checked against each UI_ITEM's *value* field. If the item's *value* field is the same as *flagSetting,* that item is marked as selected.

- *item*$_{in}$ is an array of UI_ITEM structures that are used to construct a set of string items within the horizontal list. For more information regarding the use of the UI_ITEM structure, see "Chapter 18—UI_ITEM" in *Programmer's Reference Volume 1.*

Example

```
#include <ui_win.hpp>

ExampleFunction1(UI_WINDOW_MANAGER *windowManager)
{
    // Create the list field.
    UIW_HZ_LIST * list = new UIW_HZ_LIST(10, 1, 42, 6, 14, 1);
    *list
        + new UIW_STRING(0, 0, 14, "Item 1", 64, STF_NO_FLAGS)
        + new UIW_STRING(0, 0, 14, "Item 2", 64, STF_NO_FLAGS)
        + new UIW_STRING(0, 0, 14, "Item 3", 64, STF_NO_FLAGS)
        + new UIW_STRING(0, 0, 14, "Item 4", 64, STF_NO_FLAGS);

    // Attach the list to the window.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 60, 10);
    *window
        + UIW_BORDER
        + new UIW_TITLE("Sample list")
        + new UIW_PROMPT(2, 1, "List:")
        + list;
    *windowManager + window;



    // The list will automatically be destroyed when the window
    // is destroyed.
}

ExampleFunction2(UI_WINDOW_MANAGER *windowManager)
{
    UI_ITEM listItems[] =
    {
        { 11,   NULL,   "Item 1.1", STF_NO_FLAGS },
        { 12,   NULL,   "Item 1.2", STF_NO_FLAGS },
        { 21,   NULL,   "Item 2.1", STF_NO_FLAGS },
        { 22,   NULL,   "Item 2.2", STF_NO_FLAGS },
        { 0,    NULL,   NULL, 0 }
    };

    // Create the window.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + UIW_BORDER
        + new UIW_TITLE(" Sample list ")
        + new UIW_PROMPT(2, 1, "List:")
        + new UIW_HZ_LIST(10, 1, 20, 6, NULL, WOF_NO_FLAGS, listltems);
    *windowManager + window;



    // The list will automatically be destroyed when the window
    // is destroyed.
}
```

UIW_HZ_LIST::~UIW_HZ_LIST

Syntax

```
#include <ui_win.hpp>

virtual ~UIW_HZ_LIST(void);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_HZ_LIST object. All objects attached to the horizontal list will also be destroyed.

UIW_HZ_LIST::Add

Syntax

```
#include <ui_win.hpp>

UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);
    or
UIW_HZ_LIST &operator + (UI_WINDOW_OBJECT *object);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function is used to add an object to the horizontal list.

- *returnValue*$_{out}$ is a pointer to *object* if the addition was successful. Otherwise, *returnValue* is NULL.

- *object*$_{in}$ is a pointer to the object to be added to the horizontal list.

The <u>second</u> overloaded operator adds an item to the UIW_HZ_LIST. This operator overload is equivalent to calling the Add() function, except that it allows the chaining of item additions to the UIW_HZ_LIST.

- *returnValue*$_{out}$ is a pointer to the UIW_HZ_LIST object. This pointer is returned so that the operator may be used in a statement containing other operations.

- *object*$_{in}$ is a pointer to the item that is to be added to the list.

UIW_HZ_LIST::ClassName

Syntax

#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to *_className.*

UIW_HZ_LIST::Destroy

Syntax

```
#include <ui_win.hpp>

virtual void Destroy (void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function destroys all the objects attached to the horizontal list.

UIW_HZ_LIST::Event

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the horizontal list object. It is declared virtual so that any derived horizontal list class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the horizontal list object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event():**

  L_BEGIN_SELECT—Indicates that the end-user began the selection of the object by pressing the mouse button down while on the object.

  L_BOTTOM—Scrolls the list to the last page and makes the last item in the list current. This message is interpreted from a keyboard event.

  L_CONTINUE_SELECT—Indicates that the end-user previously clicked down on the object with the mouse and is now continuing to hold the mouse button down while on the object.

  L_DOUBLE_CLICK—Indicates that the end-user double-clicked on an object with the mouse.

  L_DOWN—Moves the focus down one object. If the current object is at the bottom of a column focus will move to the top item in the next column. If the current object is at the end of the list and the WNF_NO_WRAP flag is not set, focus will move to the first item in the list. This message is interpreted from a keyboard event.

  L_END_SELECT—Indicates that the selection process, initiated with the L_-BEGIN_SELECT message, is complete. For example, the end-user has pressed and released the mouse button.

  L_LEFT—Moves the focus left one object. This message is interpreted from a keyboard event.

  L_NEXT—The list object processes this message by suppressing it. This allows the list's parent window to process it. This message is interpreted from a keyboard event.

  L_PGDN—Causes the list to scroll right a page. This message is interpreted from a keyboard event.

L_PGUP—Causes the list to scroll left a page. This message is interpreted from a keyboard event.

L_PREVIOUS—The list object processes this message by suppressing it. This allows the list's parent window to process it. This message is interpreted from a keyboard event.

L_RIGHT—Moves the focus right one object. This message is interpreted from a keyboard event.

L_SELECT—Indicates that an object on the list has been selected.

L_TOP—Scrolls the list to the first page and makes the first item in the list current. This message is interpreted from a keyboard event.

L_UP—Moves the focus up one object. If the current object is at the top of a column focus will move to the bottom item in the previous column. If the current object is at the beginning of the list and the WNF_NO_WRAP flag is not set, focus will move to the last item in the list. This message is interpreted from a keyboard event.

L_VIEW—Indicates that the mouse is being moved over the list. This message allows the list to alter the mouse image.

S_ADD_OBJECT—Causes a new object to be added to the list, *event.data* will point to the new object to be added.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_CURRENT—Causes the object to draw itself to appear current. This message is sent by the Window Manager to the window when it becomes current. The window, in turn, passes this message to the object on the window that is current. The horizontal list passes the message to its current item.

S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager sends this message to a window when the window is subtracted from the Window Manager. The window, in turn, relays the message to all objects attached to it. The horizontal list sends the message to all its children.

S_DISPLAY_ACTIVE—Causes the object to draw itself to appear active. An active object is one that is on the active (i.e., current) window. Most objects do not display differently whether they are active or inactive. An active object should not be confused with a current object. An object is active if it is on the active window. However, it may not be the current object on the window.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed by the event overlaps the region of the object.

S_DISPLAY_INACTIVE—Causes the object to draw itself to appear inactive. An inactive object is one that is not on the active (i.e., current) window. Most objects do not display differently whether they are inactive or active.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed with the event overlaps the region of the object.

S_DRAG_COPY_OBJECT—Indicates the user is dragging the object to copy it.

S_DRAG_MOVE_OBJECT—Indicates the user is dragging the object to move it.

S_DROP_COPY_OBJECT—Indicates the user dropped an object to copy it to this object.

S_DROP_MOVE_OBJECT—Indicates the user dropped an object to move it to this object.

S_HSCROLL—Causes the list to scroll horizontally, *event.scroll.delta* indicates how far to scroll.

S_HSCROLL_CHECK—Causes the list to scroll the current item into view if it is not currently visible.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When the window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_MOVE—Causes the object to update its location. The distance to move is contained in the *position* field of UI_EVENT. For example, an *event.position.-line* of -10 and an *event.position.column* of 15 moves the object 10 lines up and 15 columns to the right.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another object or window.

S_REDISPLAY—Causes the object to redraw.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

S_SUBTRACT_OBJECT—Causes an object to be subtracted from the list. *event.data* will point to the object to be subtracted.

All other events are passed by **Event()** to **UIW_WINDOW::Event()** for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event( )** function.

UIW_HZ_LIST::Information

Syntax

```
#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST request, void *data,
    ZIL_OBJECTID objectID = ID_DEFAULT);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- $returnValue_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- $request_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the horizontal list:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_DESTROY_LIST—Destroys all non-support objects attached to the list. This request simply calls **Destroy( ).**

  I_GET_BITMAP_ARRAY—Returns a pointer to the bitmap array of the current object if it has a bitmap. If a bitmap does not exist, NULL is returned. If this message is sent, *data* must be a pointer to **ZIL_UINT8.**

  I_GET_TEXT—Returns a pointer to the text associated with the current object. If this request is sent, *data* should be a doubly-indirected pointer to **ZIL_ICHAR.** This request does not copy

  I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_BITMAP_ARRAY—Sets the bitmap array associated with the current object, if it has a bitmap. If this message is sent, *data* must be a pointer to an array of **ZIL_UINT8** that contains the object's new bitmap.

All other requests are passed by **Information()** to **UIW_WINDOW::Information()** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include <ui_win.hpp>
#include <string.h>

ExampleFunction()
{
    UIW_HZ_LIST *list;



    WOF_FLAGS flags;
    list->Information(I_GET_FLAGS, &flags, ID_WINDOW_OBJECT);



    flags = WOF_BORDER;
    list->Information(I_SET_FLAGS, &flags, ID_WINDOW_OBJECT);
    flags = WOF_NON_SELECTABLE;
    list->Information(I_CLEAR_FLAGS, &flags, ID_WINDOW_OBJECT);
```

UIW_HZ_LIST::ScrollEvent

Syntax

#include <ui_win.hpp>

virtual EVENT_TYPE ScrollEvent(UI_EVENT *&event);*

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function handles events related to scrolling the horizontal list. Any events that may result in the list's scroll region getting updated (e.g., S_CREATE, L_SIZE) will call this function to update the scroll information. This function is used by OpenZinc. The programmer typically will not call this function.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time scrolling message for the horizontal list object. The type of operation performed depends on the event. The following logical events are processed by **ScrollEvent():**

  S_SCROLLRANGE—Updates the scroll values maintained in *scroll* and *vScrollInfo.* This event also updates the scroll bar's information, if one exists.

  S_HSCROLL_CHECK—Causes the list to scroll the current item into view if it is not currently visible.

  S_HSCROLL_WINDOW—Causes the objects on the horizontal list to scroll horizontally, *event.scroll.delta* should contain the amount to scroll.

UIW_HZ_LIST::Sort

Syntax

```
#include <ui_gen.hpp>

void Sort(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function sorts the UIW_HZ_LIST object using the *compareFunction* that was assigned in the constructor. If the list has no compare function, no sort occurs.

UIW_HZ_LIST::Subtract
UIW_HZ_LIST::operator -

Syntax

```
#include <ui_gen.hpp>

UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT *object);
    or
UIW_HZ_LIST &operator - (UI_WINDOW_OBJECT *object);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These functions remove an object from the UIW_HZ_LIST.

The first function removes an object from the UIW_HZ_LIST but does not call the destructor associated with the object. The programmer is responsible for deletion of each object explicitly subtracted from a list.

- *returnValue*$_{out}$ is a pointer to the next item in the list. This value is NULL if there are no more items after the subtracted item.

- *element*$_{in}$ is a pointer to the item to be subtracted from the list.

The second overloaded operator removes an item from the UIW_HZ_LIST but does not call the destructor associated with the object. This operator overload is equivalent to calling the Subtract() function, except that it allows the chaining of item removals from the UIW_HZ_LIST.

- *returnValue*$_{out}$ is a pointer to the UIW_HZ_LIST object. This pointer is returned so that the operator may be used in a statement containing other operations.

- *object*$_{in}$ is a pointer to the item that is to be subtracted from the list.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_HZ_LIST::UIW_HZ_LIST

Syntax

```
#include <ui_win.hpp>

UIW_HZ_LIST(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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_HZ_LIST by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a horizontal list is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_HZ_LIST::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_HZ_LIST from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_HZ_LIST::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- *name*$_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UIJVINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW HZ LIST::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New()** function.

- $returnValue_{out}$ is a pointer to the object's **New()** function.

UIW_HZ_LIST::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- $name_{in}$ is the name of the object to be stored.

- $file_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see

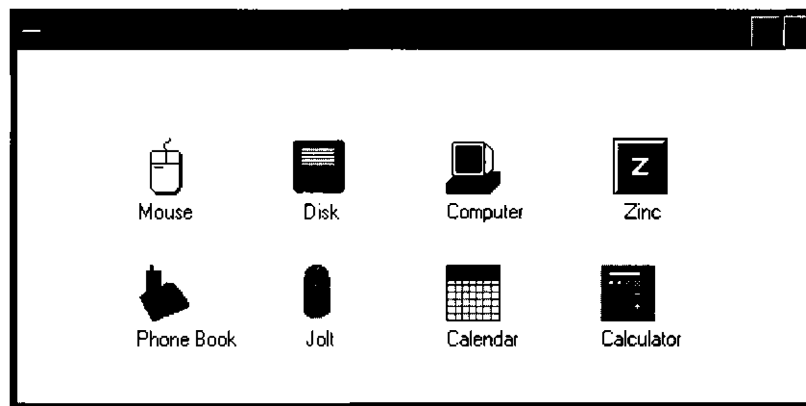
the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 9 - UIW_ICON

The UIW_ICON class can be used in several ways: to perform an action when selected by the end-user; to provide an image to enhance an application's appearance or to clarify an operation; or as a minimize icon for a window. An icon displays a 32x32 pixel image (in graphics modes only). In text mode, only the icon's text, if any, will be displayed.

Icons used within OpenZinc Application Framework may be created using the image editor in OpenZinc Designer, or they may be converted from an operating system-specific icon image using the OpenZinc Designer. The figure below shows the graphic implementation of several UIW_ICON objects:



The UIW_ICON class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_ICON : public UI_WINDOW_OBJECT
{
public:
    static ZIL_ICHAR _className[];
    static ZIL_ICHAR _applicationIconName[];
    static ZIL_ICHAR _asteriskIconName[];
    static ZIL_ICHAR _exclamationIconName[];
    static ZIL_ICHAR _handIconName[];
    static ZIL_ICHAR _questionIconName[];
    ICF_FLAGS icFlags;

    UIW_ICON(int left, int top, ZIL_ICHAR *iconName,
        ZIL_ICHAR *title = ZIL_NULLP(ZIL_ICHAR),
        ICF_FLAGS icFlags = ICF_NO_FLAGS,
        WOF_FLAGS woFlags = WOF_JUSTIFY_CENTER | WOF_NON_SELECTABLE,
        ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION));
    virtual ~UIW_ICON(void);
    virtual ZIL_ICHAR *ClassName(void);
    ZIL_ICHAR *DataGet(void);
    void DataSet(ZIL_ICHAR *text);
```

```
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

#if defined (ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_ICON(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
#if defined (ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM  *userTable);
#endif

protected:
    ZIL_ICHAR  *tit1e;
    ZIL_ICHAR  *iconName;
    int  iconWidth;
    int  iconHeight;
    ZIL_UINT8  *iconArray;
    ZIL_ICON_HANDLE  icon;
#if defined(ZIL_MSDOS) | | defined(ZIL_CURSES)
    UI_REGION  iconRegion;
    UI_REGION  titleRegion;
#endif

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

General Members

This section describes those members that are used for general purposes.

_className contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_ICON class, *_className* is "UIW_ICON."

_applicationIconName is the name of the "application" icon in the default OpenZinc data file. The application icon is the icon used as the minimize icon. *_applicationlcon-Name* is "APPLICATION" by default.

- *_asteriskIconName* is the name of the "asterisk" icon in the default OpenZinc data file. The asterisk icon is the icon used when presenting information to the user. *_asterisk-IconName* is "ASTERISK" by default.

- *_exclamationIconName* is the name of the "exclamation" icon in the default OpenZinc data file. The exclamation icon is the icon used to alert the user to a situation. *_exclamationIconName* is "EXCLAMATION" by default.

- *_handlconName* is the name of the "hand" icon in the default OpenZinc data file. The hand icon is the icon used to warn the user of a dangerous condition. *Jiandlcon-Name* is "HAND" by default.

- *_questionIconName* is the name of the "question" icon in the default OpenZinc data file. The question icon is the icon used as the minimize icon for the help system. *_questionIconName* is "QUESTION" by default.

- *icFlags* are flags that define the operation of the UIW_ICON class. A full description of the icon flags is given in the UIW_ICON constructor.

- *title* is the text that is shown on the icon.

- *iconName* is the icon's name in a OpenZinc **.DAT** file or an operating system resource file. *iconName* is used if the icon is to be read from a OpenZinc **.DAT** file or from an operating system resource file. When the icon is to be loaded, OpenZinc Application Framework first searches the operating system resources for the icon image. If the icon was not found as an operating system resource, *UI_WINDOW_OBJECT::-defaultStorage* is searched.

- *iconWidth* is the width of the icon bitmap.

- *iconHeight* is the height of the icon bitmap.

- *iconArray* is an array of **ZILJJINT8** that contains the actual bitmap data.

- *icon* is an operating system-specific icon handle. Wherever possible, the icon array is converted to a format that the operating system or graphics library can process more efficiently than a bitmap array, *icon* is a handle to the converted icon.

- *iconRegion* is the region occupied by the icon image. This member is available in DOS and Curses only.

- *titleRegion* contains the region occupied by the icon's title string (if any). This member is available in DOS and Curses only.

UIW_ICON::UIW_ICON

Syntax

```
#include <ui_win.hpp>

UIW_ICON(int left, int top, ZIL_ICHAR *iconName,
    ZIL_ICHAR *title = ZIL_NULLP(ZIL_ICHAR),
    ICF_FLAGS icFlags = ICF_NO_FLAGS,
    WOF_FLAGS woFlags = WOF JUSTIFY_CENTER | WOF_NON_SELECTABLE,
    ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_ICON class object.

- *left*$_{in}$ and *top*$_{in}$ is the starting position of the icon field within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- *iconName*$_{jn}$ is the name of the icon as it is stored in the .DAT or resource file.

- *title*$_{in}$ is the text to place directly under the icon when it is displayed on the screen.

- *icFlags*$_{in}$ are flags that define the operation of the UIW_ICON class. The following flags (declared in UI_GEN.HPP) control the general presentation of a UIW_ICON class object:

  ICF_DOUBLE_CLICK—Causes the *userFunction* function, if one exists, to be called if the user double-clicks on the icon.

ICF_MINIMIZE_OBJECT—Indicates that the icon is to be used as the minimize icon for a window.

ICF_NO_FLAGS—Does not associate any special flags with the UIW_ICON class object. In this case the icon requires a down and up click from the mouse to complete an action. This flag is set by default in the constructor.

ICF_STATICJCONARRAY—Causes the bitmap array that is used for the image to not be deleted. By default, when an icon is created, the icon image array is converted to a native storage structure and the array is deleted. If the image array should not be deleted after this conversion is performed (e.g., if the same icon image is to be used for multiple objects), then the ICF_STATIC_-ICONARRAY flag should be set.

- *woFlags*$_{in}$ are flags (common to all window objects) that determine the general operation of the icon object. The following flags (declared in **UI_WIN.HPP)** control the general presentation of, and interaction with, a UIW_ICON class object:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOF_JUSTIFY_CENTER—Center-justifies the text within the displayed icon. This flag is set by default in the constructor.

  WOF_JUSTIFY_RIGHT—Right-justifies the text within the displayed icon.

  WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is 1/10th the size of a normal cell by default.

  WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

WOF_NO_FLAGS—Does not associate any special window flags with the object. Setting this flag left-justifies the data. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor select the icon. This flag is set by default in the constructor.

- $userFunction_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. The user function will be called by the library when:

  1—the user moves onto the field,

  2—the <ENTER> key is pressed while the icon is current, if the mouse is clicked on the object or, if the icon has the ICF_DOUBLE_CLICK flag set, the user double-clicks on the icon,

  3—the user moves to a different field in the window or to a different window.

  Because the user function is called at these times, the programmer can do data validation or any other type of necessary operation. The definition of the *userFunction* is as follows:

  EVENT_TYPE FunctionName(UI_WINDOW_OBJECT **object,*
  UI_EVENT *&event,* EVENT_TYPE *ccode);*

  $returnValue_{out}$ indicates if an error has occurred. *returnValue* should be 0 if the no error occurred. Otherwise, the programmer should call the error system with an appropriate error message and return -1.

  $object_{in}$ is a pointer to the object for which the user function is being called. This argument must be typecast by the programmer if class-specific members need to be accessed.

  $event_{in}$ is the run-time message passed to the object.

$ccode_{in}$ is the logical or system code that caused the user function to be called. This code (declared in UI_EVT.HPP) will be one of the following constant values:

L_DOUBLE_CLICK—The icon has the ICF_DOUBLE_CLICK flag set and the user double-clicked on the icon.

L_SELECT—The <ENTER> key was pressed while the field was current, or the button was clicked on with the mouse.

S_CURRENT—The object just received focus because the user moved to it from another field or window. This code is sent before any editing operations are permitted.

S_NON_CURRENT—The object just lost focus because the user moved to another field or window.

Example

```
#include <ui_win.hpp>

main()
{

    UI_WINDOW_OBJECT::defaultStorage = new ZIL_STORAGE("resource.dat");

    // Attach the icon to a window.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_BORDER
        + new UIW_MAXIMIZE_BUTTON
        + new UIW_MINIMIZE_BUTTON
        + new UIW_SYSTEM_BUTTON
        + new UIW_TITLE("Example Icons", WOF_JUSTIFY_CENTER)
        + new UIW_ICON(0, 0, "minlcon", NULL, ICF_MINIMIZE_OBJECT)
        + new UIW_ICON(7, 3, "iconLogo", "Box Logo");
    *windowManager + window;

    delete UI_WINDOW_OBJECT::defaultStorage;

}
```

UIW_ICON::~UIW_ICON

Syntax

```
#include <ui_win.hpp>

virtual ~UIW_ICON(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_ICON object.

UIW_ICON::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to *_className.*

UIW_ICON::DataGet

Syntax

#include <ui_win.hpp>

ZIL_ICHAR *DataGet(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function is used to return the text information associated with the icon object.

- *returnValue*$_{out}$ is a pointer to the icon's text string.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_ICON *iconObject)
{
    ZIL_ICHAR *text = iconObject->DataGet();



}
```

UIW_ICON::DataSet

Syntax

#include <ui_win.hpp>

void DataSet(ZIL_ICHAR *"text);*

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function is used to set the text information associated with the icon object.

- $text_{in}$ is a pointer to the new text string.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_ICON *icon)
{
    ZIL_ICHAR text[] = *file";
    icon->DataSet(text);
}
```

UIW_ICON::DrawItem

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This virtual advanced function is used to draw the object. If the WOS_OWNERDRAW status is set for the object, this function will be called when drawing the icon. This allows the programmer to derive a new class from UIW_ICON and handle the drawing of the icon, if desired.

- $returnValue_{out}$ is a response based on the success of the function call. If the object is drawn the function returns a non-zero value. If the object is not drawn, 0 is returned.

- $event_{in}$ contains the run-time message that caused the object to be redrawn. *event.region* contains the region in need of updating. The following logical events may be sent to the **DrawItem()** function:

  S_CURRENT, S_NON_CURRENT, S_DISPLAY_ACTIVE and **S_DISPLAY_INACTIVE**—Messages that cause the object to be redrawn.

  WM_DRAWITEM—A message that causes the object to be redrawn. This message is specific to Windows and OS/2.

  Expose—A message that causes the object to be redrawn. This message is specific to Motif.

- $ccode_{in}$ contains the logical interpretation of *event.*

UIW_ICON::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the icon object. It is declared virtual so that any derived icon class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the icon object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event( ):**

  L_BEGIN_SELECT—Indicates that the end-user began the selection of the object by pressing the mouse button down while on the object.

  L_DOUBLE_CLICK—Indicates that the object was double-clicked. If the ICF_DOUBLE_CLICK flag is set, the icon's user function will be called.

  L_END_SELECT—Indicates that the selection process, initiated with the L_-BEGIN_SELECT message, is complete. For example, the end-user has pressed and released the mouse button. The user function will be called.

  L_SELECT—Indicates that the object has been selected. The selection may be the result of a mouse click or a keyboard action.

  S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

  S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it

to cause the window and all the objects attached to the window to determine their positions.

S_CURRENT—Informs the object that it has become current. Typically, the object draws itself to appear current. This message is sent by the Window Manager to a window when it becomes current. The window, in turn, passes this message to the object on the window that is current.

S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager sends this message to a window when the window is subtracted from the Window Manager. The window, in turn, relays the message to all objects attached to it.

S_DISPLAY_ACTIVE—Causes the object to draw itself to appear active. An active object is one that is on the active (i.e., current) window. Most objects do not display differently whether they are active or inactive. An active object should not be confused with a current object. An object is active if it is on the active window. However, it may not be the current object on the window.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed by the event overlaps the region of the object. This message is sent by a UIW_WINDOW to all the non-current, active objects attached to it.

S_DISPLAY_INACTIVE—Causes the object to draw itself to appear inactive. An inactive object is one that is not on the active (i.e., current) window. Most objects do not display differently whether they are inactive or active.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed with the event overlaps the region of the object. This message is sent by a UIW_WINDOW to all the objects attached to it.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another field or window.

S_REDISPLAY—Causes the object to redraw.

S_REGION_DEFINE—Causes the object to reserve a region of the screen in which it will display.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

S_RESET_DISPLAY—Changes the display to a different resolution, *event.data* should point to the new display class to be used. If *event.data* is NULL, a text mode display will be created. This event is specific to DOS and must be placed on the event queue by the programmer. The library will never generate this event.

All other events are passed by Event() to UI_WINDOW_OBJECT::Event() for processing.

UIW_ICON::Information

Syntax

```
#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST request, void *data,
    ZIL_OBJECTID objectID = ID_DEFAULT);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the icon:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_CLEAR_FLAGS—Clears the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be cleared, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to clear the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_-OBJECT. If the ICF_FLAGS are to be cleared, *objectID* should be ID_ICON. This allows the object to process the request at the proper level. This request only clears those flags that are passed in; it does not simply clear the entire field.

  I_COPY_TEXT—Copies the *text* associated with the object into a buffer provided by the programmer. If this request is sent, *data* must be the address of a buffer where the string's text will be copied. This buffer must be large enough to contain all of the characters associated with the button and the terminating NULL character.

  I_GET_FLAGS—Requests the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to obtain the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the ICF_FLAGS are desired, *objectID* should be ID_ICON. This allows the object to process the request at the proper level.

  I_GET_ICON_ARRAY—Returns a pointer to the icon's image array. If an icon image does not exist, NULL is returned. If this message is sent, *data* must be a pointer to a variable of type **ZIL_UINT8.** This request does not copy the image into a new buffer.

I_GET_TEXT—Returns a pointer to the text associated with the object. If this request is sent, *data* should be a doubly-indirected pointer to **ZIL_ICHAR.** This request does not copy the text into a new buffer.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_ICON_ARRAY—Sets the icon image array associated with the icon. If this message is sent, *data* must be a pointer to an array of **ZIL_UINT8** that contains the icon's new image.

I_SET_FLAGS—Sets the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be set, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to set the WOF_-FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the ICF_-FLAGS are to be set, *objectID* should be ID_ICON. This allows the object to process the request at the proper level. This request only sets those flags that are passed in; it does not clear any flags that are already set.

I_SET_TEXT—Sets the text associated with the object. This request will also redisplay the object with the new text, *data* should be a pointer to the new text.

All other requests are passed by **Information()** to **UI_WINDOW_OBJECT::-Information( )** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include <ui_win.hpp>

ExampleFunction()
{
    UIW_ICON *icon, *icon1, *icon2;

    char string[30];
    icon->Information(I_COPY_TEXT, string);
    icon1->Information(I_SET_TEXT, "Clock");
    icon2->Information(I_SET_TEXT, "Phone Book");



}
```

Storage Members

This section describes those class members that are used for storage purposes.

UIW_ICON::UIW_ICON

Syntax

#include <ui_win.hpp>

UIW_ICON(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));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_ICON by loading the object from a data file. Typically, the programmer does not need to use this constructor. If an icon is stored

in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- *$name_{in}$* is the name of the object to be loaded.

- *$file_{in}$* is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *$object_{in}$* is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *$objectTable_{in}$* is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *$userTable_{in}$* is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_ICON::Load

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_ICON from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_ICON::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UIJWIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_ICON::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's New() function.

- $returnValue_{out}$ is a pointer to the object's New() function.

UIW_ICON::Store

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- $name_{in}$ is the name of the object to be stored.

- $file_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

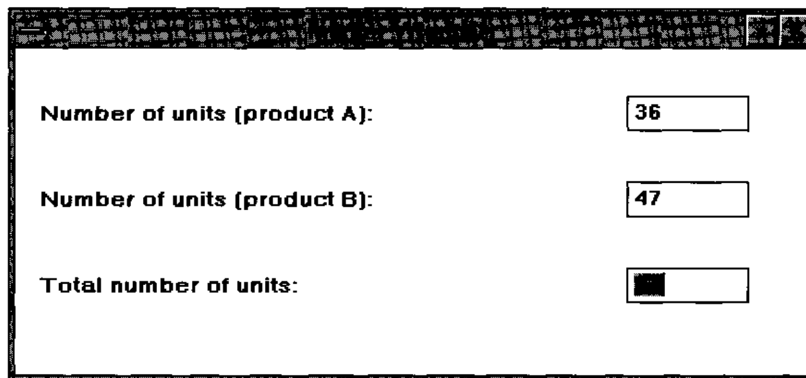
- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the

description of *UI_WINDOW_0BJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 10 - UIW_INTEGER

The UIW_INTEGER class is used to display numeric information and to collect information, in integer form, from an end-user. The UIW_INTEGER class supports integer values only. If larger values are required or if any formatting is necessary (e.g., currency symbols) the UIW_BIGNUM object should be used. The figure below shows the graphical implementation of a window with several variations of the UIW_INTEGER class object:



The UIW_INTEGER class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_INTEGER : public UIW_STRING
{
public:
    static ZIL_ICHAR _className[];
    static int defaultlnitialized;
    NMF_FLAGS nmFlags;

    UIW_INTEGER(int left, int top, int width, int *value,
        const ZIL_ICHAR *range = ZIL_NULLP(ZIL_ICHAR),
        NMF_FLAGS nmFlags = NMF_NO_FLAGS,
        WOF_FLAGS woFlags = WOF_BORDER | WOF_AUTO_CLEAR,
        ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION));
    virtual ~UIW_INTEGER(void);
    virtual ZIL_ICHAR *ClassName(void);
    int DataGet(void);
    void DataSet(int *value);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);
    virtual int Validatefint processError = TRUE);

#if defined (ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void) ;
    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));
    UIW_INTEGER(const ZIL_ICHAR *name, ZIL_3T0RAGE_READ_0NLY *flle,
        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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

    void SetLanguage(const ZIL_ICHAR *languageName);

protected:
    int *number;
    ZIL_ICHAR *range;
    const ZIL__LANGUAGE *myLanguage;
} ;
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_INTEGER class, *_className* is "UIW_INTEGER."

- *defaultlnitialized* indicates if the default language strings for this object have been set up. The default strings are located in the file **LANG J)EF.CPP.** If *defaultlnitialized* is TRUE, the strings have been set up. Otherwise they have not been.

- *nmFlags* are flags that define the operation of the UIW_INTEGER class. A full description of the number flags is given in the UIW_INTEGER constructor.

- *number* is used to store the integer value for UIW_INTEGER. If the WOF_NO_-ALLOCATE_DATA flag is set, *number* will simply point to the value that was passed in the constructor.

- *range* is a string that specifies the range(s) of acceptable integer values, *range* is a copy of the range that is passed to the constructor.

- *myLanguage* is the ZIL_LANGUAGE object that contains the string translations for this object.

UIW_INTEGER::UIW_INTEGER

Syntax

#include <ui_win.hpp>

UIW_INTEGER(int *left,* int *top,* int *width,* int **value,*
  const ZIL_ICHAR **range* = ZIL_NULLP(ZIL_ICHAR),
  NMF_FLAGS *nmFlags* = NMF_NO_FLAGS,
  WOF_FLAGS *woFlags* = WOF_BORDER | WOF_AUTO_CLEAR,
  ZIL_USER_FUNCTION *userFunction* = ZIL_NULLF(ZIL_USER_FUNCTION));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_INTEGER class object.

- $left_{in}$ and top_{in} is the starting position of the integer field within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the integer field. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value. The height of the integer field is determined automatically by the UIW_INTEGER object.

- $value_{in}$ is a pointer to the default numeric value. This value is copied into a buffer allocated by the UIW_INTEGER object unless the WOF_NO_ALLOCATE_DATA flag is set, in which case *value* is used.

- $range_{in}$ is a string that specifies the valid numeric ranges. A range consists of a minimum value, a maximum value, and the values in between. For example, if a range of "1000..10000" is specified, the UIW_INTEGER class object will only accept those numeric values that fall between 1,000 and 10,000, inclusive. Open-ended ranges can be specified by leaving the minimum or maximum value off. For

example, a range of "500.." will allow all values that are 500 or greater. Multiple, disjoint ranges can be specified by separating the individual ranges with a slash (i.e. '/'). For example, "100..199/1000.." will accept all values from 100 to 199 and values of 1000 or greater. If *range* is NULL, any number within the absolute range is accepted. This string is copied by the UIW_INTEGER class object to the *range* member variable.

- *nmFlags*$_{in}$ describes how the integer should display and interpret the numeric information. The following flags (declared in **UI_GEN.HPP)** control the general presentation of a UIW_INTEGER class object:

  NMF_NO_FLAGS—Does not associate any special flags with the number object. This flag should not be used in conjunction with any other NMF flag. This is the default argument in the constructor.

- *woFlags*$_{in}$ are flags (common to all window objects) that determine the general operation of the integer object. The following flags (declared in **UI_WIN.HPP)** affect the operation of a UIW_INTEGER class object:

  WOF_AUTO_CLEAR—Automatically marks the entire buffer if the end-user tabs to the field from another object. If the user then enters data (without first having pressed any movement or editing keys) the entire field will be replaced. This flag is set by default in the constructor.

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOFJNVALID—Sets the initial status of the field to be "invalid." Invalid entries fit in the absolute range determined by the object type but do not fulfill all the requirements specified by the program. For example, an integer may initially be set to 200, but the final number, edited by the end-user, must be in the range "10.. 100." The initial number in this example fits the absolute range requirements of a UIW_INTEGER class object but does not fit into the specified range. By denoting the field as invalid, you force the user to enter an acceptable value.

  WOF_JUSTIFY_CENTER—Center-justifies the data within the displayed field.

WOF_JUSTIFY_RIGHT—Right-justifies the data within the displayed field.

WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is 1/10th the size of a normal cell by default.

WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

WOF_NO_FLAGS—Does not associate any special window flags with the object. Setting this flag left-justifies the data. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor edit the integer information. Typically, the field will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

WOF_SUPPORT_OBJECT—Causes the object to be placed in the parent object's support list. The support list is reserved for objects that are not displayed as part of the user region of the window. Care should be used when setting this flag on an object that does not use it by default as undesirable effects may occur. This flag generally should not be used by the programmer.

WOF_UNANSWERED—Sets the initial status of the field to be "unanswered." An unanswered field is displayed as an empty field.

WOF_VIEW_ONLY—Prevents the object from being edited. However, the object may become current and the user may scroll through the data, mark it, and copy it.

- $userFunction_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. The user function will be called by the library when:

1—the user moves onto the field,

2—the <ENTER> key is pressed while the field is current or, if the field is in a list, the mouse is clicked on it, or

3—the user moves to a different field in the window or to a different window.

Because the user function is called at these times, the programmer can do data validation or any other type of necessary operation. The definition of the *userFunction* is as follows:

```
EVENT_TYPE FunctionName(UI_WINDOW_OBJECT *object,
    UI_EVENT &event, EVENT_TYPE ccode);
```

$returnValue_{out}$ indicates if an error has occurred. *returnValue* should be 0 if the no error occurred. Otherwise, the programmer should call the error system with an appropriate error message and return -1.

$object_{in}$ is a pointer to the object for which the user function is being called. This argument must be typecast by the programmer if class-specific members need to be accessed.

$event_{in}$ is the run-time message passed to the object.

$ccode_{in}$ is the logical or system code that caused the user function to be called. This code (declared in **UI_EVT.HPP)** will be one of the following constant values:

L_SELECT—The <ENTER> key was pressed while the field was current, or, if the field is in a list, the mouse was clicked on the field.

S_CURRENT—The object just received focus because the user moved to it from another field or window. This code is sent before any editing operations are permitted.

S_NON_CURRENT—The object just lost focus because the user moved to another field or window.

NOTE: If a user function is associated with the object, **Validate(**) must be called explicitly from within *userFunction* if range checking is desired.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Create a window and add it to the window manager,
    int iValue = 0;
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_BORDER
        + new UIW_TITLE(" Sample numbers ")
        + new UIW_PROMPT(2, 1, "Integer:")
        + new UIW_INTEGER(12, 1, 20, &iValue, "0. .10000");
    *windowManager + window;



    // The integer will automatically be destroyed when the window
    // is destroyed.
}
```

UIW_INTEGER::~UIW_INTEGER

Syntax

#include <ui_win.hpp>

virtual ~UIW_INTEGER(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_INTEGER object.

Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
```

```
    int number = 100;

    UIW_INTEGER *integer = new UIW_INTEGER(1, 1, 20, &number);

    delete integer;
}
```

UIW_INTEGER::ClassName

Syntax

#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- $returnValue_{out}$ is a pointer to *_className.*

UIW_INTEGER::DataGet

Syntax

#include <ui_win.hpp>

int DataGet(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function gets the current numeric information associated with the UIW_INTEGER class object.

- *returnValue*$_{out}$ is the integer value.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_INTEGER *integerObject)
{
    int value = integerObject->DataGet();


}
```

UIW_INTEGER::DataSet

Syntax

#include <ui_win.hpp>

void DataSet(int *"value);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function assigns a new value to the UIW_INTEGER object and redisplays the field. If no value is passed in (i.e., *value* is NULL), the field will be redrawn.

- $value_{in}$ is a pointer to the new value. If the WOF_NO_ALLOCATE_DATA flag is set, this argument must be an integer, allocated by the programmer, that is not destroyed until the UIW_INTEGER class object is destroyed. Otherwise, the information associated with this argument is copied by the UIW_INTEGER class object. If this argument is NULL, no numeric information is changed, but the number field is redisplayed.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_INTEGER *number)
{
    int amount = 100;
    number->DataSet(fcamount);
}
```

UIW_INTEGER::Event

Syntax

#include <ui_win.hpp>

virtual EVENT_TYPE Event(const UI_EVENT *See verity,*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the integer object. It is declared virtual

so that any derived integer class can override its default operation.

- *returnValue*$_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the integer object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event( ):**

  L_SELECT—Indicates that the object has been selected. The selection may be the result of a mouse click or a keyboard action.

  S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

  S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another field or window.

  All other events are passed by **Event()** to **UIW_STRING::Event()** for processing.

UIW_INTEGER::Information

Syntax

```
#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST request, void *data,
    ZIL_OBJECTID objectID = ID_DEFAULT);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc objects and programmer functions to get or modify specified information about an object.

- $returnValue_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- $request_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the integer:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_DECREMENT_VALUE—Decrements the integer's value. If this message is sent, *data* must be a pointer to an integer. The integer object's value will be decremented by the value of *data.* The integer will not be modified if the new value is not within the specified range.

  I_GET_VALUE—Returns the *value* associated with the integer. If this message is sent, *data* must be a pointer to a variable of type **int** where the integer's value will be copied.

  I_INCREMENT_VALUE—Increments the integer's value. If this message is sent, *data* must be a pointer to an integer. The integer object's value will be incremented by the value of *data.* The integer will not be modified if the new value is not within the specified range.

  I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

  I_SET_VALUE—Sets the *value* associated with the integer. If this message is sent, *data* must be a pointer to a variable of type **int** that contains the integer's new value.

  All other requests are passed by **Information( )** to **UIW_STRING::Information( )** for processing.

- *data*$_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- *objectID*$_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information( )** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

UIW_INTEGER::SetLanguage

Syntax

```
#include <ui_win.hpp>

void SetLanguage(const ZIL_ICHAR *languageName);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function sets the language to be used by the object. The string translations for the object will be loaded and the object's *myLanguage* member will be updated to point to the new ZIL_LANGUAGE object. By default, the object uses the language identified in the **LANG_DEF.CPP** file, which compiles into the library. (If a different default language is desired, simply copy a **LANG_<ISO>.CPP** file from the OpenZinc\SOURCE\-INTL directory to the \OpenZinc\SOURCE directory, and rename it to **LANG_DEF.CPP** before compiling the library.) The language translations are loaded from the **I18N.DAT** file, so it must be shipped with your application.

- *languageName*$_{in}$ is the two-letter ISO language code identifying which language the object should use.

UIW_INTEGER::Validate

Syntax

#include <ui_win.hpp>

virtual int Validate(int *processError* = TRUE);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function is used to validate objects. When an object receives the S_CURRENT or S_NON_CURRENT messages, it calls **Validate( )** to check if the value entered is valid. However, if a user function is associated with the object, **Validate()** must be called explicitly from the user function if range checking is desired. The value is invalid if it is not within the absolute range of the object or if it is not within a range specified by the *range* member variable.

- *returnValue*$_{out}$ indicates the result of the validation. The possible values for *returnValue* are:

  NMI_GREATER_THAN_RANGE—The number entered was greater than the maximum value of a negatively open-ended range.

  NMI_INVALID—The number was entered in an incorrect format.

  NMI_LESS_THAN_RANGE—The number entered was less than the minimum value of a positively open-ended range.

  NMI_OK—The number was entered in a correct format and within the valid ranges.

  NMI_OUT_OF_RANGE—The number was not within the valid range for numbers or was not within the specified *range.*

• *$processError_{in}$* determines whether **Validate()** should call **UI_ERROR_SYSTEM::-ReportError()** if an error occurs. If *processError* is TRUE, **ReportError( )** is called. Otherwise, the error system is not called.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_INTEGER::UIW_INTEGER

Syntax

```
#include <ui_win.hpp>

UIW_INTEGER(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));
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced constructor creates a new UIW_INTEGER by loading the object from a data file. Typically, the programmer does not need to use this constructor. If an integer is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

• *$name_{in}$* is the name of the object to be loaded.

• *$file_{in}$* is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_INTEGER::Load

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_INTEGER from a persistent object data

file. It is called by the persistent constructor and is typically not used by the programmer.

- *name*$_{in}$ is the name of the object to be loaded.

- file$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- object$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_INTEGER::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_INTEGER::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New()** function.

- *returnValue*$_{out}$ is a pointer to the object's **New( )** function.

UIW_INTEGER::Store

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

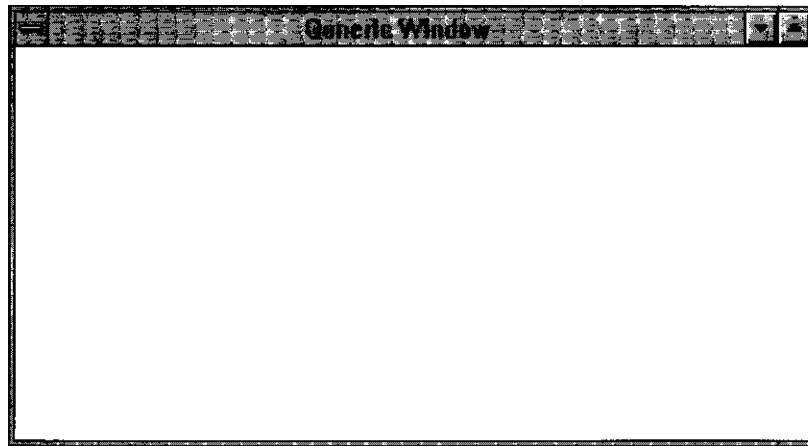
- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 11 - UIW_MAXIMIZE_BUTTON

The UIW_MAXIMIZE_BUTTON class is used to maximize a window. A maximized window fills the whole screen if it is attached to the Window Manager. If a window is an MDI child window, it fills the client area of its MDI parent window when maximized. When a window has been maximized, the maximize button becomes a restore button. If the restore button is selected, the window will return to its normal size. The figure below shows a graphical implementation of a window with a UIW_MAXIMIZE_BUTTON class object (the button with the character):



NOTE: The Macintosh does not allow maximizing, as such. Instead, the Macintosh has a zoom box, which allows the end-user to toggle the window size between two different sizes. The zoom box is created in place of the maximize button in a Macintosh application built using OpenZinc.

The UIW_MAXIMIZE_BUTTON class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_MAXIMIZE_BUTTON : public UIW_BUTTON
{
public:
    static ZIL_ICHAR _className[];
    static int defaultlnitialized;

    UIW_MAXIMIZE_BUTTON(void);
    virtual ~UIW_MAXIMIZE_BUTTON(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

#if defined(ZIL_LOAD)
```

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

    void SetDecorations(const ZIL_ICHAR *decorationName);

protected:
    const ZIL_DECORATION *myDecorations;
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_MAXIMIZE_BUTTON class, *_className* is "UIW_MAXIMIZE_BUTTON."

- *defaultlnitialized* indicates if the default decorations (i.e., images) for this object have been set up. The default decorations are located in the file **IMG_DEF.CPP.** If *defaultlnitialized* is TRUE, the decorations have been set up. Otherwise they have not been.

- *myDecorations* is the ZIL_DECORATION object that contains the images for this object.

UIW_MAXIMIZE_BUTTON::UIW_MAXIMIZE_BUTTON

Syntax

#include <ui_win.hpp>

UIW_MAXIMIZE_BUTTON(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_MAXIMIZE_BUTTON class object. The maximize button object is always positioned in the upper right corner of the parent window. To ensure that the maximize button is drawn correctly, it must be added right after the UIW_-BORDER class object. The following example shows the correct order of maximize button addition.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Create a window and attach it to the window manager.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_BORDER
        + new UIW_MAXIMIZE_BUTTON
        + new UIW_MINIMIZE_BUTTON
        + new UIW_SYSTEM_BUTTON
        + new UIW_TITLE("Window 1");
    *windowManager + window;



    // The maximize button will automatically be destroyed when the window
    // is destroyed.
}
```

UIW_MAXIMIZE_BUTTON::~UIW_MAXIMIZE_BUTTON

Syntax

```
#include <ui_win.hpp>

virtual ~UIW_MAXIMIZE_BUTTON(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_MAXIMIZE_BUTTON object.

UIW_MAXIMIZE_BUTTON::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to *_className.*

UIW_MAXIMIZE_BUTTON::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the maximize button object. It is declared virtual so that any derived maximize button class can override its default operation.

- *returnValue*$_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the maximize button object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event( ):**

  L_BEGIN_SELECT—Indicates that the end-user pressed the mouse button down. This begins the selection process of an object. This event is interpreted from an event generated by the mouse device.

  L_CONTINUE_SELECT—Indicates that the end-user previously clicked down on the object with the mouse and is now continuing to hold the mouse button down while on the object.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

All other events are passed by **Event()** to **UIW_BUTTON::Event()** for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event()** function.

UIW_MAXIMIZE_BUTTON::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request*, void **data,*
ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- $returnValue_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- $request_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the maximize button:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

  All other requests are passed by **Information( )** to **UIW_BUTTON::Information( )** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information( )** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID*, the object will attempt to interpret the request with the *objectID* of the actual object type.

UIW_MAXIMIZE_BUTTON::SetDecorations

Syntax

```
#include <ui_win.hpp>

void SetDecorations(const ZIL_ICHAR *decorationName);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function sets the decorations (i.e., images) to be used by the object. The images for the object will be loaded and the object's *myDecorations* member will be updated to point to the new ZIL_DECORATION object. By default, the object uses the images identified in the **IMG_DEF.CPP** file, which compiles into the library. (If different default images are desired, simply copy a **IMG_<ISO>.CPP** file from the OpenZinc\SOURCE\INTL directory to the \OpenZinc\SOURCE directory, and rename it to **IMG_DEF.CPP** before compiling the library.) The images are loaded from the **I18N.DAT** file, so it must be shipped with your application.

- *decorationName*$_{in}$ is the two-letter ISO country code identifying which images the object should use.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_MAXIMIZE_BUTTON::UIW_MAXIMIZE_BUTTON

Syntax

```
#include <ui_win.hpp>

UIW_MAXIMIZE_BUTTON(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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_MAXIMIZE J3UTTON by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a maximize button is stored in a data file it is usually stored as part of a UIW_-WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_W1NDOW JOB JECT:-.objectTable* in "Chapter 43—UI_WIN-

DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_MAXIMIZE_BUTTON::Load

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_MAXIMIZE_BUTTON from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *UIJWINDOWJOB JECT:-.userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_MAXIMIZE_BUTTON::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.-objectTable* in "Chapter 43—UI.WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_MAXIMIZE_BUTTON::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New()** function.

returnValue$_{out}$ is a pointer to the object's **New()** function.

UIW_MAXIMIZE_BUTTON::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

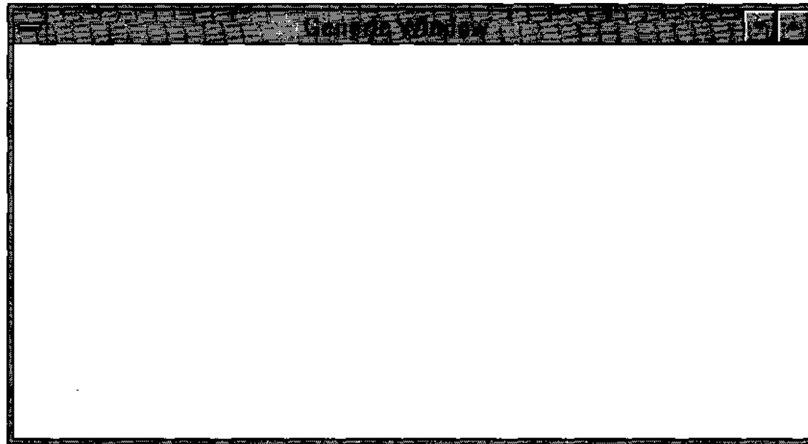
- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *U1JVINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 12 - UIW_MINIMIZE_BUTTON

The UIW_MINIMIZE_BUTTON class is used to minimize a window. If an icon with the ICF_MINIMIZE_OBJECT flag set has been added to the window, the window is reduced to that icon when the minimize button is selected. The figure below shows a graphical implementation of a window with a UIW_MINIMIZE_BUTTON object (the button with the 'T' character):



NOTE: The Macintosh does not allow the minimizing of windows. Thus, OpenZinc will ignore this object and it will have no effect if used in a Macintosh application.

The UIW_MINIMIZE_BUTTON class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_MINIMIZE_BUTTON : public UIW_BUTTON
{
public:
    static ZIL_ICHAR _className[];
    static int defaultInitialized;

    UIW_MINIMIZE_BUTTON(void);
    virtual ~UIW_MINIMIZE_BUTTON(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_MINIMIZE_BUTTON(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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE__OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

    void SetDecorations(const ZIL_ICHAR *decorationName);

protected:
    const ZIL_DECORATION *myDecorations;
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_MINIMIZE_BUTTON class, *_className* is "UIW_MINIMIZE_BUTTON."

- *defaultInitialized* indicates if the default decorations (i.e., images) for this object have been set up. The default decorations are located in the file **IMG_DEF.CPP.** If *defaultInitialized* is TRUE, the decorations have been set up. Otherwise they have not been.

- *myDecorations* is the ZIL_DECORATION object that contains the images for this object.

UIW_MINIMIZE_BUTTON::UIW_MINIMIZE_BUTTON

Syntax

#include <ui_win.hpp>

UIW_MINIMIZE_BUTTON(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_MINIMIZE_BUTTON class object. The minimize button object is always positioned in the upper right corner of the parent window. To ensure that the minimize button is drawn correctly, it must be added right after the UIW_-MAXIMIZE_BUTTON class object. The following example shows the correct order of minimize button addition.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Create a window and attach it to the window manager.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_BORDER
        + new UIW_MAXIMIZE_BUTTON
        + new UIW_MINIMIZEJBUTTON
        + new UIW_SYSTEM_BUTTON
        + new UIW_TITLE("Window 1");
    *windowManager + window;



    // The minimize button will automatically be destroyed when the window
    //is destroyed.
}
```

UIW_MINIMIZE_BUTTON::~UIW_MINIMIZE_BUTTON

Syntax

#include <ui_win.hpp>

virtual ~UIW_MINIMIZE_BUTTON(void);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_MINIMIZE_BUTTON object.

UIW_MINIMIZE_BUTTON::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to *_className.*

UIW_MINIMIZE_BUTTON::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the minimize button object. It is declared virtual so that any derived minimize button class can override its default operation.

- $returnValue_{in}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the minimize button object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event():**

  L_BEGIN_SELECT—Indicates that the end-user pressed the mouse button down. This begins the selection process of an object. This event is interpreted from an event generated by the mouse device.

  L_CONTINUE_SELECT—Indicates that the end-user previously clicked down on the object with the mouse and is now continuing to hold the mouse button down while on the object.

  S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

All other events are passed by **Event()** to **UIWJBUTTON::Event( )** for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event()** function.

UIW_MINIMIZE_BUTTON::Information

Syntax

```
#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST request, void *data,
    ZIL_OBJECTID objectID = ID_DEFAULT);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the minimize button:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

  All other requests are passed by **Information**() to **UIW_BUTTON::Information()** for processing.

- *data*$_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- *objectID*$_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information**() function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

UIW_MINIMIZE_BUTTON::SetDecorations

Syntax

#include <ui_win.hpp>

void SetDecorations(const ZIL_ICHAR **decorationName);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function sets the decorations (i.e., images) to be used by the object. The images for the object will be loaded and the object's *myDecorations* member will be updated to point to the new ZIL_DECORATION object. By default, the object uses the images identified in the **IMG_DEF.CPP** file, which compiles into the library. (If different default images are desired, simply copy a **IMG_<ISO>.CPP** file from the OpenZinc\SOURCE\INTL directory to the \OpenZinc\SOURCE directory, and rename it to **IMG_DEF.CPP** before compiling the library.) The images are loaded from the **I18N.DAT** file, so it must be shipped with your application.

- $decorationName_{in}$ is the two-letter ISO country code identifying which images the object should use.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_MINIMIZE_BUTTON::UIW_MINIMIZE_BUTTON

Syntax

```
#include <ui_win.hpp>

UIW_MINIMIZE_BUTTON(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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_MINIMIZE_BUTTON by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a minimize button is stored in a data file it is usually stored as part of a UIW_-WINDOW and will be loaded when the window is loaded.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW MINIMIZE_BUTTON::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This <u>advanced</u> function is used to load a UIW_MINIMIZE_BUTTON from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UIJWIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_MINIMIZE_BUTTON::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- name$_{in}$ is the name of the object to be loaded.

- *$file_{in}$* is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *$object_{in}$* is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *$userTable_{in}$* is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *UI_WINDOWJDBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_MINIMIZE_BUTTON::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's New( ) function.

- *returnValue*$_{out}$ is a pointer to the object's New( ) function.

UIW_MINIMIZE_BUTTON::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

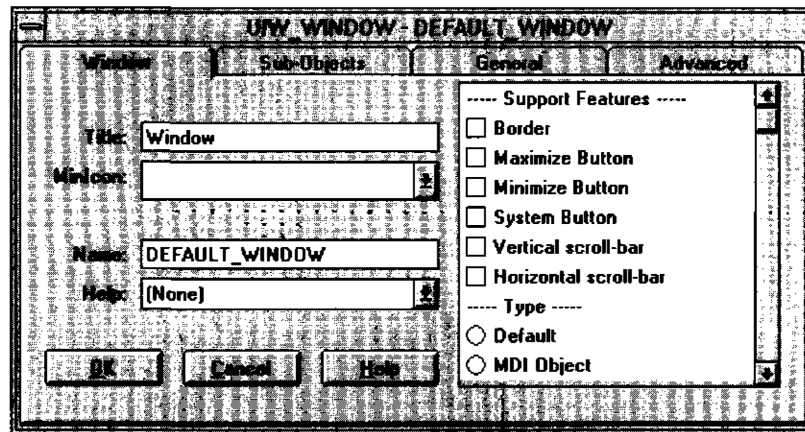
- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see

the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 13 - UIW_NOTEBOOK

The UIW_NOTEBOOK class is used to display multiple pages of related objects. The class is so named because it's appearance resembles a notebook with tabs on the pages. Each tab, when selected with the mouse, will "turn" to that page. The tabs contain text identifying the page's contents. The figure below shows a graphical implementation of a window with a UIW_NOTEBOOK object:



The UIW_NOTEBOOK class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_NOTEBOOK : public UIW_WINDOW
{
public:
    static int borderWidth;
    static int shadowWidth;
    static ZIL_ICHAR _className[];

    UIW_NOTEBOOK(void);
    ~UIW_NOTEBOOK(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE DrawItem(const UI_EVENT &event, EVENT_TYPE ccode);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(INFO_REQUEST request, void *data,
        OBJECTID objectID = ID_DEFAULT);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_NOTEBOOK(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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

    // List members.
    UIW_WINDOW *Add(UIW_WINDOW *object);
    UIW_NOTEBOOK &operator+(UIW_WINDOW *object);
    }
```

General Members

This section describes those members that are used for general purposes.

- *borderWidth* is the width of the border that appears around the notebook edges.

- *shadowWidth* is the width of the shadow that appears around the notebook edges.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_NOTEBOOK class, *_className* is "UIW_NOTEBOOK."

UIW_NOTEBOOK::UIW_NOTEBOOK

Syntax

#include <ui_win.hpp>

UIW_NOTEBOOK(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_NOTEBOOK class object.

UIW_NOTEBOOK::~UIW_NOTEBOOK

Syntax

#include <ui_win.hpp>

virtual ~UIW_NOTEBOOK(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_NOTEBOOK object. All objects attached to the notebook will also be destroyed.

UIW_NOTEBOOK::Add
UIW_NOTEBOOK::operator +

Syntax

#include <ui_win.hpp>

UIW_WINDOW *Add(UIW_WINDOW **object);*
or
UIW_NOTEBOOK &operator + (UIW_WINDOW **object);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These overloaded functions are used to add a page of objects to the notebook. Each page of objects on the notebook is created by adding a UIW_WINDOW, with sub-objects attached to it. The order in which the windows are added determines the order in which the pages will appear. The first window added will be the first, or front, page. The title bar text is used to identify the page on the tab.

The first function adds a page to the UIW_NOTEBOOK.

- $returnValue_{out}$ is a pointer to *object* if the addition was successful. Otherwise, *returnValue* is NULL.

- $object_{in}$ is a pointer to the window to be added to the notebook.

The second overloaded operator adds a page to the UIW_NOTEBOOK. This operator overload is equivalent to calling the **UIW_NOTEBOOK::Add()** function except that it allows the chaining of object additions to the UIW_NOTEBOOK.

- $returnValue_{out}$ is a pointer to the UIW_NOTEBOOK. This pointer is returned so that the operator may be used in a statement containing other operations.

- $object_{in}$ is a pointer to the window that is to be added to the notebook.

UIW_NOTEBOOK::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to *_className.*

UIW_NOTEBOOK::DrawItem

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual advanced function is used to draw the- object. If the WOS_OWNERDRAW status is set for the object, this function will be called when drawing the notebook. This allows the programmer to derive a new class from UIW_NOTEBOOK and handle the drawing of the notebook, if desired.

- *returnValue*$_{out}$ is a response based on the success of the function call. If the object is drawn the function returns a non-zero value. If the object is not drawn, 0 is returned.

- $event_{in}$ contains the run-time message that caused the object to be redrawn. *event.region* contains the region in need of updating. The following logical events may be sent to the **DrawItem()** function:

  S_CURRENT, S_NON_CURRENT, S_DISPLAY_ACTIVE and **S_DISPLAY_INACTIVE**—Messages that cause the object to be redrawn.

  WM_DRAWITEM—A message that causes the object to be redrawn. This message is specific to Windows and OS/2.

  Expose—A message that causes the object to be redrawn. This message is specific to Motif.

- $ccode_{in}$ contains the logical interpretation of *event.*

UIW_NOTEBOOK::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function processes run-time messages sent to the notebook object. It is declared virtual so that any derived notebook class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *$event_{in}$* contains a run-time message for the notebook object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event():**

  S_CHANGED—Causes the object to recalculate its position and size. When a notebook is moved or sized, the objects on the notebook will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

  S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a notebook is attached to it to cause the notebook and all the objects attached to the notebook to determine their positions.

  S_SIZE—Causes the object to recalculate its position and size. When a notebook is sized, the objects on the notebook will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

  S_CHANGE_PAGE—Causes the notebook to "turn" to a new page. The page number that should be turned to is subtracted from S_CHANGE_PAGE and passed as the event type. The first page that was added is page zero. For example, if ten pages were added, and the application needs to turn to page seven, an event with a type of S_CHANGE_PAGE - 6 should be sent to the notebook. Thus, an S_CHANGE_PAGE event by itself will turn to the first page in the notebook.

  All other events are passed by **Event()** to **UIW_WINDOW::Event()** for processing.

  NOTE: Because some graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event()** function.

UIW_NOTEBOOK::Information

Syntax

```
#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST request, void *data,
    ZIL_OBJECTID objectID = ID_DEFAULT);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the notebook:

  I_GET_NUMBERID_OBJECT—Returns a pointer to an object whose *numberlD* matches the value in *data,* if one exists. This object does a depth-first search of the objects attached to it, looking for a match of the *numberlD.* If no object has a *numberlD* that matches *data,* NULL is returned. If this message is sent, *data* must be a pointer to a NUMBERID. Programmers should use a window's *numberlD* with caution as it may change at run-time. For more details, see the note accompanying the description of **UI_WINDOW_-OBJECT::NumberID()** in "Chapter 43—UI_WINDOW_OBJECT" of *Programmer's Reference Volume 1.*

  I_GET_STRINGID_OBJECT—Returns a pointer to an object whose *stringlD* matches the character string in *data,* if one exists. This object does a depth-first search of the objects attached to it looking for a match of the *stringlD.* If no

object has a *stringID* that matches *data,* NULL is returned. If this message is sent, *data* must be a pointer to a string.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_SEARCH_PAGE—Sets the page to be searched on a subsequent I_GET_NUMBERID_OBJECT or I_GET_STRINGID_OBJECT request. If the notebook has many pages and the number of the page containing the desired object is known, sending the I_SET_SEARCH_PAGE request will speed up the subsequent object request. If the I_SET_SEARCH_PAGE request is sent, *data* must be a pointer to an int that contains the page number to be searched. If *data* is -1, then subsequent I_GET_NUMBERID_OBJECT and I_GET_STRINGID_-OBJECT requests will start with the first page and continue through each page until the object is found.

All other requests are passed by **Information()** to **UIW_WINDOW::Information()** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information( )** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_NOTEBOOK::UIW_NOTEBOOK

Syntax

```
#include <ui_win.hpp>

UIW_NOTEBOOK(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));
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This advanced constructor creates a new UIW_NOTEBOOK by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a maximize button is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-

DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_NOTEBOOK::Load

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_NOTEBOOK from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT: .userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_NOTEBOOK::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_NOTEBOOK::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New()** function.

- *returnValue*$_{out}$ is a pointer to the object's **New()** function.

UIW_NOTEBOOK::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

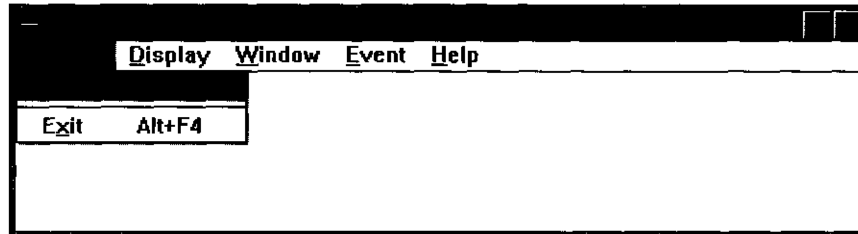
- *$object_{in}$* is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- *$objectTable_{in}$* is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *$userTable_{in}$* is a pointer to a table that contains the addresses of user objects, user functions, and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 14 - UIW_POP_UP_ITEM

The UIW_POP_UP_ITEM class is used to display menu options in a pop-up menu. A pop-up item can have a sub-menu associated with it. The figure below shows a graphical implementation of the UIW_POP_UP_ITEM objects (shown as "Refresh" and "Exit" on the pop-up menu):



The UIW_POP_UP_ITEM class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_POP_UP_ITEM : public UIW_BUTTON
{
public:
    static ZIL_ICHAR _className[];
    static int defaultlnitialized;
    MNIF_FLAGS mniFlags;
    UIW_POP_UP_MENU menu;

    UIW_POP_UP_ITEM(void);
    UIW_POP_UP_ITEM(ZIL_ICHAR *text, MNIF_FLAGS mniFlags = MNIF_NO_FLAGS,
        BTF_FLAGS btFlags = BTF_N0_3D, WOF_FLAGS woFlags = WOF_NO_FLAGS,
        ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION),
        EVENT_TYPE value = 0);
    UIW_POP_UP_ITEM(int left, int top, int width, ZIL_ICHAR *text,
        MNIF_FLAGS mniFlags = MNIF_NO_FLAGS,
        BTF_FLAGS btFlags = BTF_NO_FLAGS,
        WOF_FLAGS woFlags = WOF_NO_FLAGS,
        ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION),
        EVENT_TYPE value = 0);
    virtual ~UIW_POP_UP_ITEM(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_POP_UP_ITEM(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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

    // List members.
    UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);
    UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT *object);
    UIW_POP_UP_ITEM &operator+(UI_WINDOW_OBJECT *object);
    UIW_POP_UP_ITEM &operator-(CJI_WINDOW_OBJECT *object);

    void SetDecorations(const ZIL_ICHAR *decorationName);

protected:
    virtual EVENT_TYPE DrawItem(const UI_EVENT &event, EVENT_TYPE ccode);
    const ZIL_DECORATION *rnyDecorations;
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_POP_UP_ITEM class, *_className* is "UIW_POP_UP_ITEM."

- *defaultlnitialized* indicates if the default decorations (i.e., images) for this object have been set up. The default decorations are located in the file **IMG_DEF.CPP.** If *defaultlnitialized* is TRUE, the decorations have been set up. Otherwise they have not been.

- *mniFlags* are flags that define the operation of the UIW_POP_UP_ITEM class. A full description of the pop-up item flags is given in the UIW_POP_UP_ITEM constructor.

- *menu* is the UIW_POP_UP_MENU that maintains the pop-up menu options that are displayed when the pop-up item is selected if it has a sub-menu.

- *myDecorations* is the ZIL_DECORATIONS object that contains the images for this object.

UIW_POP_UP_ITEM::UIW_POP_UP_ITEM

Syntax

```
#include <ui_win.hpp>

UIW_POP_UP_ITEM(void);
    or
UIW_POP_UP_ITEM(ZIL_ICHAR *text, MNIF_FLAGS mniFlags = MNIF_NO_FLAGS,
    BTF_FLAGS btFlags = BTF_N0_3D,W0F_FLAGS woFlags = WOF_NO_FLAGS,
    ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION),
    EVENT_TYPE value = 0);
    or
UIW_POP_UP_ITEM(int left, int top, int width, ZIL_ICHAR *text,
    MNIF_FLAGS mniFlags = MNIF_NO_FLAGS,
    BTF_FLAGS btFlags = BTF_NO_FLAGS,
    WOF_FLAGS woFlags = WOF_NO_FLAGS,
    ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION),
    EVENT_TYPE value = 0);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These overloaded constructors create a new UIW_POP_UP_ITEM class object.

The first constructor takes no arguments. It creates a menu separator that appears as a horizontal line when placed in a pop-up menu.

The second constructor creates a pop-up item. It takes the following arguments:

- *text*$_{in}$ is the text that is shown in the pop-up item. A hotkey for the pop-up item may be specified by inserting the '&' character into the string before the desired hotkey character. For example, if the string "Exit" is to be displayed and 'x' is to be the hotkey, the string should be entered as "E&xit." The '&' will not be displayed, but will cause the hotkey character to be drawn appropriately. If an '&' is required in

the text that is displayed, place two '&' characters in the string (e.g., "A && B" will display as "A & B" and the pop-up item will not have a hotkey). This string is copied by the UIW_POP_UP_ITEM class unless the WOF_NO_ALLOCATE_-DATA flag is set. If this flag is set, *text* must be space, allocated by the programmer, that is not deleted until the UIW_POP_UP_ITEM object has been deleted.

- *mniFlags*$_{in}$ are flags that define the operation of the UIW_POP_UP_ITEM class. The following flags (declared in **UI_WIN.HPP)** control the general presentation of a UIW_POP_UP_ITEM class object:

  MNIF_CHECK_MARK—Causes the pop-up item to display a check mark at the front of the text when the pop-up item is selected.

  MNIF_NO_FLAGS—Does not associate any special flags with the pop-up item. This flag should not be used in conjunction with any other MNIF flags. This flag is set by default in the constructor.

  MNIF_SEPARATOR—Causes the pop-up item to be a separator that will appear as a horizontal line when placed in a pop-up menu.

  MNIF_SEND_MESSAGE—Causes an event to be created and put on the event queue when the menu item is selected. The pop-up item's *value* is used as the *EVENT_TYPE* of the event placed on the queue. If this flag is set, the pop-up item should not have a user function. This flag is equivalent to the BTF_SEND_-MESSAGE flag.

- *btFlags*$_{in}$ are flags that define the operation of the UIW_POP_UP_ITEM class. The following flags (declared in **UI_WIN.HPP)** control the general presentation of a UIW_POP_UP_ITEM class object:

  BTF_DOWN_CLICK—Completes the item's action on a mouse down-click, rather than on a down-click and release action.

  BTF_NO_FLAGS—Does not associate any special flags with the UIW_-POP_UP_ITEM class object. In this case the item requires a down and up click from the mouse to complete an action. This flag should not be used in conjunction with any other BTF flags.

  BTF_NO_TOGGLE—Causes the pop-up item to not toggle between selected and non-selected states. This flag should not be set if the MNIF_CHECK_-MARK flag is set.

BTF_SEND_MESSAGE—Causes an event to be created and put on the event queue when the menu item is selected. The pop-up item's *value* is used as the *EVENT_TYPE* of the event placed on the queue. If this flag is set, the pop-up item should not have a user function. This flag is equivalent to the MNIF_SEND_-MESSAGE flag.

- *woFlags*$_{in}$ are flags (common to all window objects) that determine the general operation of the window object. The following flags (declared in UI_WIN.HPP) control the general presentation of, and interaction with, a pop-up item:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOF_JUSTIFY_CENTER—Center-justifies the text within the displayed button.

  WOF_JUSTIFY_RIGHT—Right-justifies the text within the displayed button.

  WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

  WOF_NO_FLAGS—Does not associate any special window flags with the object. Setting this flag left-justifies the data. This flag should not be used in conjunction with any other WOF flags.

  WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor select the pop-up item. Typically, the object will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

- *userFunction*$_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. The user function will be called by the library when:

  1—the user moves onto the field,

2—the <ENTER> key is pressed while the pop-up item is current or the mouse is clicked on the object, or

3—the user moves to a different field in the window or to a different window.

Because the user function is called at these times, the programmer can do data validation or any other type of necessary operation. The definition of the *userFunction* is as follows:

```
EVENT_TYPE FunctionName(UI_WINDOW_OBJECT *object,
    UI_EVENT &event, EVENT_TYPE ccode);
```

returnValue$_{out}$ indicates if an error has occurred. *returnValue* should be 0 if the no error occurred. Otherwise, the programmer should call the error system with an appropriate error message and return -1.

object$_{in}$ is a pointer to the object for which the user function is being called. This argument must be typecast by the programmer if class-specific members need to be accessed.

event$_{in}$ is the run-time message passed to the object.

ccode$_{in}$ is the logical or system code that caused the user function to be called. This code (declared in **UI_EVT.HPP)** will be one of the following constant values:

L_SELECT—The <ENTER> key was pressed while the field was current, or the pop-up item was clicked on with the mouse.

S_CURRENT—The object just received focus because the user moved to it from another field or window.

S_NON_CURRENT—The object just lost focus because the user moved to another field or window.

- *value*$_{in}$ is an event that will be placed on the event queue if the pop-up item has the MNIF_SEND_MESSAGE flag set and the end-user selects the pop-up item, *value* can also be used as an identifier to distinguish between several pop-up items in a common user function. For example, the programmer could associate the value 0 with a "Save" menu item and a value of 1 with a "Save As" menu item. This allows the programmer to define one user function that can determine the action to take based on the pop-up item's value.

The <u>third</u> constructor creates a pop-up item. It takes the following arguments:

- *left*$_{in}$ and *top*$_{in}$ is the starting position of the pop-up item within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- *width*$_{in}$ is the width of the pop-up item. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value. The height of the button is determined automatically by the UIW_-POP_UP_ITEM object.

- *text*$_{in}$ is the text that is shown in the pop-up item. A hotkey for the pop-up item may be specified by inserting the '&' character into the string before the desired hotkey character. For example, if the string "Exit" is to be displayed and 'x' is to be the hotkey, the string should be entered as "E&xit." The '&' will not be displayed, but will cause the hotkey character to be drawn appropriately. If an '&' is required in the text that is displayed, place two '&' characters in the string (e.g., "A && B" will display as "A & B" and the pop-up item will not have a hotkey). This string is copied by the UIW_POP_UP_ITEM class unless the WOF_NO_ALLOCATE_-DATA flag is set. If this flag is set, *text* must be space, allocated by the programmer, that is not deleted until the UIW_POP_UP_ITEM object has been deleted.

- *mniFlags*$_{in}$ are flags that define the operation of the UIW_POP_UP_ITEM class. For a description of valid MNIF_FLAGS, see the description of the <u>second</u> constructor, above.

- *btFlags*$_{in}$ are flags that define the operation of the UIW_POP_UP_ITEM class. For a description of valid BTF_FLAGS, see the description of the <u>second</u> constructor, above.

- *woFlags*$_{in}$ are flags (common to all window objects) that determine the general operation of the window object. For a description of valid WOF_FLAGS, see the description of the <u>second</u> constructor, above.

- *userFunction*$_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. For a description of how the user function is used, see the description of the <u>second</u> constructor, above.

- *value*$_{in}$ is an event that will be placed on the event queue if the pop-up item has the MNIF_SEND_MESSAGE flag set and the end-user selects the pop-up item, *value* can also be used as an identifier to distinguish between several pop-up items in a

common user function. For example, the programmer could associate the value 0 with a "Save" menu item and a value of 1 with a "Save As" menu item. This allows the programmer to define one user function that can determine the action to take based on the pop-up item's value.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Create a window with basic window objects.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_BORDER
        + new UIW_MAXIMIZE_BUTTON
        + new UIW_MINIMIZE_BUTTON
        + &(*new UIW_SYSTEM_BUTTON
            + new UIW_POP_UP_ITEM(""Restore", MNIF_RESTORE)
            + new UIW_POP_UP_ITEM(""Move", MNIF_MOVE)
            + new UIW_POP_UP_ITEM("~Size", MNIF_SIZE))
        + new UIW_TITLE("Window 1");

    *windowManager + window;



    // The pop-up items are automatically destroyed when the window
    // is destroyed.
}
```

UIW_POP_UP_ITEM::~UIW_POP_UP_ITEM

Syntax

#include <ui_win.hpp>

virtual ~UIW_POP_UP_ITEM(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_POP_UP_-ITEM object.

UIW_POP_UP_ITEM::Add
UIW_POP_UP_ITEM::operator +

Syntax

```
#include <ui_win.hpp>

UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);
    or
UIW_POP_UP_ITEM &operator + (UI_WINDOW_OBJECT *object);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

These overloaded functions are used to add a new pop-up item to the UIW_POP_UP_-ITEM object's sub-menu.

The first overloaded function adds a new pop-up item to the UIW_POP_UP_ITEM object.

- $returnValue_{out}$ is a pointer to *object* if the addition was successful. Otherwise, *returnValue* is NULL.

- $object_{in}$ is a pointer to the new pop-up item to be added to the sub-menu.

The second overloaded operator adds a new pop-up item to the UIW_POP_UP_ITEM object's sub-menu. This operator overload is equivalent to calling the Add( ) function except that it allows the chaining of list element additions to the UIW_POP_UP_ITEM object.

- *returnValue*$_{out}$ is a pointer to the UIW_POP_UP_ITEM object. This pointer is returned so that the operator may be used in a statement containing other operations.

- *object*$_{in}$ is a pointer to the new pop-up item that is to be added to the sub-menu.

UIW_POP_UP_ITEM::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to *_className.*

UIW_POP_UP_ITEM::DrawItem

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual advanced function is used to draw the object. If the WOS_OWNERDRAW status is set for the object, this function will be called when drawing the pop-up item. This allows the programmer to derive a new class from UIW_POP_UP_ITEM and handle the drawing of the pop-up item, if desired.

- *returnValue*$_{out}$ is a response based on the success of the function call. If the object is drawn the function returns a non-zero value. If the object is not drawn, 0 is returned.

- *event*$_{in}$ contains the run-time message that caused the object to be redrawn. *event, region* contains the region in need of updating. The following logical events may be sent to the **DrawItem()** function:

  S_CURRENT, S_NON_CURRENT, S_DISPLAY_ACTIVE and **S_DISPLAY_INACTIVE**—Messages that cause the object to be redrawn.

  WM_DRAWITEM—A message that causes the object to be redrawn. This message is specific to Windows and OS/2.

  Expose—A message that causes the object to be redrawn. This message is specific to Motif.

- *ccode*$_{in}$ contains the logical interpretation of *event.*

UIW POP_UP ITEM::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the pop-up item object. It is declared virtual so that any derived pop-up item class can override its default operation.

- *returnValue*$_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the pop-up item object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event():**

  L_END_SELECT—Indicates that the selection process, initiated with the L_-BEGIN_SELECT message, is complete. For example, the end-user has pressed and released the mouse button. The user function will be called if this event is received.

  L_LEFT—Moves to a new pop-up menu. If the current pop-up menu is a sub-menu, it is closed and its parent's menu is made current. If the current pop-up menu is not a sub-menu, the pop-up menu of the pull-down item to the left of the current pull-down item is opened. This message is interpreted from a keyboard event.

  L_RIGHT—Moves to a new pop-up menu. If the current pop-up item has a sub-menu, it is opened. If the current pop-up item does not have a sub-menu, the pop-up menu of the pull-down item to the right of the current pull-down item is opened. This message is interpreted from a keyboard event.

  L_SELECT—Indicates that the object has been selected. The selection may be the result of a mouse click or a keyboard action.

  S_ADD_OBJECT—Causes a new pop-up item to be added to the object's sub-menu. *event.data* will point to the new pop-up item to be added.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_CURRENT—Causes the object to draw itself to appear current. This message is sent by the Window Manager to a window when it becomes current. The window, in turn, passes this message to the object on the window that is current.

S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager sends this message to a window when the window is subtracted from the Window Manager. The window, in turn, relays the message to all objects attached to it.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

S_RESET_DISPLAY—Changes the display to a different resolution, *event.data* should point to the new display class to be used. If *event.data* is NULL, a text mode display will be created. This event is specific to DOS and must be placed on the event queue by the programmer. The library will never generate this event.

S_SUBTRACT_OBJECT—Causes a pop-up item to be subtracted from the object's sub-menu, *event.data* will point to the pop-up item to be subtracted.

S_VERIFY_STATUS—Causes the object to correlate its state (i.e., selected or not selected) with the operating system.

All other events are passed by Event() to UIW_BUTTON::Event( ) for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event( )** function.

UIW_POP_UP_ITEM::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request,* void *"data,*
ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- $returnValue_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- $request_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the pop-up item:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

I_CLEAR_FLAGS—Clears the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be cleared, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to clear the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_-OBJECT. If the MNIF_FLAGS are to be cleared, *objectID* should be ID_POP_-UP_ITEM. This allows the object to process the request at the proper level. This request only clears those flags that are passed in; it does not simply clear the entire field.

I_GET_FLAGS—Requests the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to obtain the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the MNIF_FLAGS are desired, *objectID* should be ID_POP_UP_ITEM. This allows the object to process the request at the proper level.

I_GET_NUMBERID_OBJECT—Returns a pointer to an object whose *numberID* matches the value in *data*, if one exists. This object does a depth-first search of the objects attached to it, looking for a match of the *numberID.* If no object has a *numberID* that matches *data,* NULL is returned. If this message is sent, *data* must be a pointer to a NUMBERID.

I_GET_STRINGID_OBJECT—Returns a pointer to an object whose *stringID* matches the character string in *data,* if one exists. This object does a depth-first search of the objects attached to it looking for a match of the *stringID.* If no object has a *stringID* that matches *data,* NULL is returned. If this message is sent, *data* must be a pointer to a string.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_FLAGS—Sets the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be set, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to set the WOF_-FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the STF_-FLAGS are to be set, *objectID* should be ID_STRING. This allows the object to process the request at the proper level. This request only sets those flags that are passed in; it does not clear any flags that are already set.

All other requests are passed by **Information()** to **UIW_BUTTON::Information()** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID*, the object will attempt to interpret the request with the *objectID* of the most derived class.

Example

```
#include <ui_win.hpp>

ExampleFunction()
{
        MNIF_FLAGS flags;
        item->Information(I_GET_FLAGS, &flags, ID_POP_UP_ITEM);


}
```

UIW_POP_UP_ITEM::SetDecorations

Syntax

#include <ui_win.hpp>

void SetDecorations(const ZIL_ICHAR **decorationName);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function sets the decorations (i.e., images) to be used by the object. The images for the object will be loaded and the object's *myDecorations* member will be updated to point to the new ZIL_DECORATION object. By default, the object uses the images identified in the **IMG_DEF.CPP** file, which compiles into the library. (If different default images are desired, simply copy a **IMG_<ISO>.CPP** file from the OpenZinc\SOURCE\INTL directory to the \OpenZinc\SOURCE directory, and rename it to **IMG_DEF.CPP** before compiling the library.) The images are loaded from the **I18N.DAT** file, so it must be shipped with your application.

- $decorationName_{in}$ is the two-letter ISO country code identifying which images the object should use.

UIW_POP_UP_ITEM::Subtract
UIW_POP_UP_ITEM::operator -

Syntax

#include <ui_win.hpp>

UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT **object);*
or

UIW_POP_UP_ITEM &operator - (UI_WINDOW_OBJECT **object);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These functions remove a pop-up item from the UIW_POP_UP_ITEM object's sub-menu.

The first function removes a pop-up item from the UIW_POP_UP_ITEM object's sub-menu but does not call the destructor associated with the object. The programmer is responsible for deletion of each object explicitly subtracted from a list.

- *returnValue*$_{out}$ is a pointer to the next pop-up item in the sub-menu. This value is NULL if there are no more pop-up items after the subtracted pop-up item.

- *object*$_{in}$ is a pointer to the pop-up item to be subtracted from the sub-menu.

The second overloaded operator removes a pop-up item from the UIW_POP_UP_ITEM object's sub-menu but does not call the destructor associated with the object. This operator overload is equivalent to calling the **Subtract^**) function, except that it allows the chaining of list element removals from the UIW_POP_UP_ITEM object.

- *returnValue*$_{out}$ is a pointer to the UIW_POP_UP_ITEM object. This pointer is returned so that the operator may be used in a statement containing other operations.

- *object*$_{in}$ is a pointer to the pop-up item that is to be subtracted from the sub-menu.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_POP_UP_ITEM::UIW_POP_UP_ITEM

Syntax

#include <ui_win.hpp>

UIW_POP_UP_ITEM(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));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_POP_UP_ITEM by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a pop-up item is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" *of Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UIJWIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT: .userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW POP_UP_ITEM::Load

Syntax

#include <ui_win.hpp>

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

UI_ITEM **userTable);*

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced function is used to load a UIW_POPJJP_ITEM from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1*.

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume* 7. If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_POP_UP_ITEM::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_POP_UP_ITEM::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's New() function.

- *returnValue*$_{out}$ is a pointer to the object's New() function.

UIW_POP_UP_ITEM::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- $name_{in}$ is the name of the object to be stored.

- $file_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

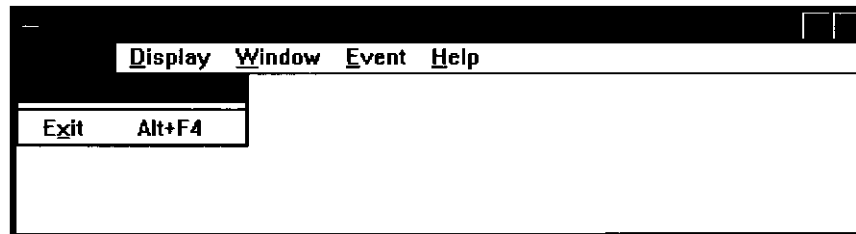
- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the

description of *UI_WINDOW_OB JECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 15 - UIW_POP_UP_MENU

The UIW_POP_UP_MENU class is used to display a menu of related option selections. A pop-up menu is typically used in connection with a pull-down menu structure, but may be used directly if desired. When used as part of the pull-down menu system, the programmer will not explicitly create a pop-up menu—the pull-down items and pop-up items manipulate the pop-up menus as needed. When using a pop-up menu as a stand-alone object, the menu options are attached to the pop-up menu as UIW_POP_UP_ITEM objects. To display a UIW_POP_UP_MENU object, it must be added to the Window Manager. The figure below shows a graphical implementation of the UIW_POP_-UP_MENU class object:



The UIW_POP_UP_MENU class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_POP_UP_MENU : public UIW_WINDOW
{
public:
    static ZIL_ICHAR _className[];

    UIW_POP_UP_MENU(int left, int top, WNF_FLAGS wnFlags,
        WOF_FLAGS woFlags = WOF_BORDER,
        WOAF_FLAGS woAdvancedFlags = WOAF_NO_FLAGS);
    UIW_POP_UP_MENU(int left, int top, WNF_FLAGS wnFlags, UI_ITEM *item);
    virtual ~UIW_POP_UP_MENU(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_POP_UP_MENU(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
	#if defined(ZIL_STORE)
		virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
			ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
			UI_ITEM *userTable);
	#endif

		// List members.
	#if defined(ZIL_MACINTOSH)
		virtual UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);
		UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT *object);
		UIW_POP_UP_MENU &operator+(UI_WINDOW_OBJECT *object);
		UIW_POP_UP_MENU &operator-(UI_WINDOW_OBJECT *object);
	#endif
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_POP_UP_MENU class, *_className* is "UIW_POP_UP_MENU."

UIW_POP_UP_MENU::UIW_POP_UP_MENU

Syntax

#include <ui_win.hpp>

UIW_POP_UP_MENU(int *left*, int *top,* WNF_FLAGS *wnFlags,*
  WOF_FLAGS *woFlags* = WOF_BORDER,
  WOAF_FLAGS *woAdvancedFlags* = WOAF_NO_FLAGS);
  or
UIW_POP_UP_MENU(int *left,* int *top,* WNF_FLAGS *wnFlags*, UI_ITEM **item);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These constructors create a new UIW_POP_UP_MENU class object.

The first constructor takes the following arguments:

- $left_{in}$ and top_{in} is the starting position of the pop-up menu. Since the pop-up menu is added to the Window Manager, this position is relative to the upper-left corner of the screen. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values. The width and height of the pop-up menu is computed automatically by the UIW_-POP_UP_MENU class object based on the size of the menu items.

- $wnFlags_{in}$ are flags that define the operation of the pop-up menu. The following flags (declared in **UI_WIN.HPP)** control the general presentation of the pop-up menu:

  WNF_AUTO_SORT—Causes the menu options to be sorted in alphabetical order.

  WNF_CONTINUE_SELECT—Allows the end-user to drag through the menu options with the mouse button pressed. If this flag is not set, the highlight on the menu options will not follow the dragging mouse. This flag should usually be set on a pop-up menu.

  WNF_NO_FLAGS—Does not associate any special flags with the pop-up menu. This flag should not be used in conjunction with any other WNF flags.

  WNF_NO_WRAP—Will not allow arrowing up or down to wrap from the end of the list to the beginning or vice versa.

  WNF_SELECT_MULTIPLE—Allows more than one option in the list to become selected at the same time. If this flag is set, the pop-up menu will still close when a selection is made, but selecting another option later will not cause the previously selected item to be un-selected. This flag is typically used if the pop-up items in the menu have the MNIF_CHECK_MARK flag set.

- $woFlags_{inm}$ are flags (common to all window objects) that determine the general operation of the pop-up menu object. The following flags (declared in **UI_WIN.HPP)** control the general presentation of, and interaction with, a UIW_-POP_UP_MENU class object:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the

graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to use the remaining available space in its parent object.

WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor select any menu items. Typically, the object will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

- *woAdvancedFlags*$_{in}$ are flags (common to all window objects) that determine the advanced operation of the pop-up menu object. The following flags (declared in UI_-WIN.HPP) control the advanced operation of a pop-up menu object:

WOAF_LOCKED—Prevents the Window Manager from removing the pop-up menu from the display. The WOAFJLOCKED flag must be cleared before the Window Manager will allow the pop-up menu to be removed from the display.

WOAF_MODAL—Prevents any other window from receiving events from the Window Manager. A modal window receives all events until it is removed from the display.

WOAF_NO_DESTROY—Prevents the Window Manager from destroying the pop-up menu. If this flag is set, the menu can be removed from the display, but the programmer is responsible for destroying the menu.

WOAF_NO_FLAGS—Does not associate any special advanced flags with the window object. This flag should not be used in conjunction with any other WOAF flags.

WOAF_NORMAL_HOT_KEYS—Allows the end-user to select an option using its hotkey by pressing the hotkey by itself, without the <Alt> key otherwise required for selecting with a hotkey.

WOAF_TEMPORARY—Causes the pop-up menu to be displayed temporarily. If another window is made current or a non-temporary window is added to the

Window Manager, all temporary windows are removed automatically by the Window Manager.

The second constructor creates a pop-up menu using a pre-defined item array. These items are used to create UIW_POP_UP_ITEM objects.

- $left_{in}$ and top_{in} is the starting position of the pop-up menu. Since the pop-up menu is added to the Window Manager, this position is relative to the upper-left corner of the screen. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values. The width and height of the pop-up menu is computed automatically by the UIW_POP_UP_-MENU class object based on the size of the menu items.

- $wnFlags_{in}$ gives information on how to display the items in the pop-up menu. The following flags (declared in **UI_WIN.HPP)** control the general presentation and operation of the pop-up menu:

  WNF_AUTO_SORT—Causes the menu options to be sorted in alphabetical order.

  WNF_CONTINUE_SELECT—Allows the end-user to drag through the menu options with the mouse button pressed. If this flag is not set, the highlight on the menu options will not follow the dragging mouse. This flag should usually be set on a pop-up menu.

  WNF_NO_FLAGS—Does not associate any special flags with the pop-up menu. This flag should not be used in conjunction with any other WNF flags.

  WNF_NO_WRAP—Prevents the current option in the pop-up menu from wrapping between the top and bottom options when arrowing through the list.

  WNF_SELECT_MULTIPLE—Allows more than one option in the list to become selected at the same time. If this flag is set, the pop-up menu will still close when a selection is made, but selecting another option later will not cause the previously selected item to be un-selected. This flag is typically used if the pop-up items in the menu have the MNIF_CHECK_MARK flag set.

- $item_{in}$ is an array of UI_ITEM structures that is used to construct a set of UIW_-POP_UP_ITEM objects within the pop-up menu. For more information regarding UI_ITEM structures, see "Chapter 18—UI_ITEM" of *Programmer's Reference Volume 1.*

UIW_POP_UP_MENU: ;~UIW_POP_UP_MENU

Syntax

#include <ui_win.hpp>

virtual ~UIW_POP_UP_MENU(void);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_POP_UP_MENU object. All objects attached to the pop-up menu will also be destroyed.

UIW_POP_UP_MENU::Add

Syntax

#include <ui_win.hpp>

UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT **object);*
or

UIW_POP_UP_MENU &operator + (UI_WINDOW_OBJECT **object);*

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function adds a new pop-up item to the UIW_POP_UP_MENU object.

- *returnValue*$_{out}$ is a pointer to *object* if the addition was successful. Otherwise, *returnValue* is NULL.

- *object*$_{in}$ is a pointer to the pop-up item to be added to the pop-up menu.

The second overloaded operator adds a pop-up item to the UIW_POP_UP_MENU object. This operator overload is equivalent to calling the Add() function, except that it allows the chaining of list element additions to the UIW_POP_UP_MENU object.

- *returnValue*$_{out}$ is a pointer to the UIW_POP_UP_MENU object. This pointer is returned so that the operator may be used in a statement containing other operations.

- *object*$_{in}$ is a pointer to the pop-up item that is to be added to the menu.

UIW_POP_UP_MENU::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to *_className.*

UIW_POP_UP_MENU::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function processes run-time messages sent to the pop-up menu object. It is declared virtual so that any derived pop-up menu class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the pop-up menu object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event():**

  L_DOWN—Moves the focus down one menu option. This message is interpreted from a keyboard event.

  L_END_SELECT—Indicates that the selection process, initiated with the L_-BEGIN_SELECT message, is complete. For example, the end-user has pressed and released the mouse button.

  L_LEFT—Moves to a new pop-up menu if the pop-up menu is associated with a pull-down menu structure. If the current pop-up menu is a sub-menu, it is closed and its parent's menu is made current. If the current pop-up menu is not a sub-menu, the pop-up menu of the pull-down item to the left of the current pull-down item is opened. This message is interpreted from a keyboard event.

L_RIGHT—Moves to a new pop-up menu if the pop-up menu is associated with a pull-down menu structure. If the current pop-up item has a sub-menu, it is opened. If the current pop-up item does not have a sub-menu, the pop-up menu of the pull-down item to the right of the current pull-down item is opened. This message is interpreted from a keyboard event.

L_UP—Moves the focus up one menu option. This message is interpreted from a keyboard event.

S_ADD_OBJECT—Causes a new pop-up item to be added to the menu. *event.data* will point to the new pop-up item to be added.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager sends this message to a window when the window is subtracted from the Window Manager. The window, in turn, relays the message to all objects attached to it.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

S_SUBTRACT_OBJECT—Causes a pop-up item to be subtracted from the menu, *event.data* will point to the pop-up item to be subtracted.

All other events are passed by **Event** to **UIW_WINDOW::Event** for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event()** function.

UIW_POP_UP_MENU: :Information

Syntax

```
#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST request, void *data,
    ZIL_OBJECTID objectID = ID_DEFAULT);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the pop-up menu:

    I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags,

particularly if the new flag settings will change the visual appearance of the object.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

All other requests are passed by **Information()** to **UIW_WINDOW::Information()** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the most derived class.

UIW_POP_UP_MENU::Subtract
UIW_POP_UP_MENU::operator -

Syntax

#include <ui_gen.hpp>

UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT **object);*
    or
UIW_POP_UP_MENU &operator - (UI_WINDOW_OBJECT **object);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These functions remove a pop-up item from the UIW_POP_UP_MENU object.

The first function removes a pop-up item from the UIW_POP_UP_MENU object but does not call the destructor associated with the object. The programmer is responsible for deletion of each object explicitly subtracted from a list.

- $returnValue_{out}$ is a pointer to the next pop-up item in the menu. This value is NULL if there are no more pop-up items after the subtracted item.

- $object_{in}$ is a pointer to the pop-up item to be subtracted from the menu.

The second overloaded operator removes a pop-up item from the UIW_POP_UP_MENU object but does not call the destructor associated with the object. This operator overload is equivalent to calling the **Subtracts**) function, except that it allows the chaining of list element removals from the UIW_POP_UP_MENU object.

- $returnValue_{out}$ is a pointer to the UIW_POP_UP_MENU object. This pointer is returned so that the operator may be used in a statement containing other operations.

- $object_{in}$ is a pointer to the pop-up item that is to be subtracted from the menu.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_POP_UP_MENU::UIW_POP_UP_MENU

Syntax

```
#include <ui_win.hpp>

UIW_POP_UP_MENU(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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_POP_UP_MENU by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a pop-up menu object is stored in a data file it is usually stored as part of a UIW_-WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UIJWIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_POP_UP_MENU::Load

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_POP_UP_MENU from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_POP_UP_MENU::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- *name*$_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1*.

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UIJWINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_POP_UP_MENU::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New( )** function.

- *returnValue*$_{out}$ is a pointer to the object's **New( )** function.

UIW_POP_UP_MENU::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

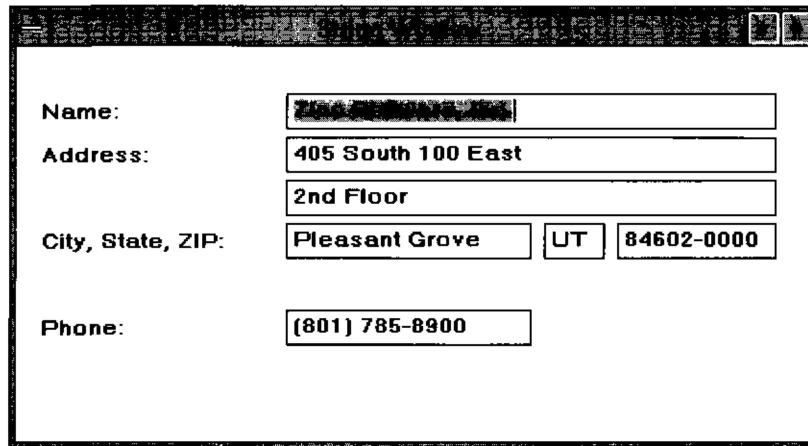
- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see

the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 16 - UIW PROMPT

The UIW_PROMPT class is used to provide lead information about another window object. A prompt cannot become current and performs no actions. The picture below shows a graphical implementation of UIW_PROMPT objects (the fields with the ':' character to their right):



The UIW_PROMPT class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_PROMPT : public UI_WINDOW_OBJECT
{
public:
    static ZIL_ICHAR _className[];

    UIW_PROMPT(int left, int top, ZIL_ICHAR *text,
        WOF_FLAGS woFlags = WOF_NO_FLAGS);
    UIW_PROMPT(int left, int top, int width, ZIL_ICHAR *text,
        WOF_FLAGS woFlags = WOF_NO_FLAGS);
    virtual ~UIW_PROMPT(void);
    virtual ZIL_ICHAR *ClassName(void);
    ZIL_ICHAR *DataGet(void);
    void DataSet(ZIL_ICHAR *text);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

#if defined (ZIL_LOAD)
    virtual ZIL_NEW_FtTNCT|ON NewFunction(void);
    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);
    UIW_PROMPT(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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

protected:
    ZIL_ICHAR *text;

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

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_PROMPT class, *_className* is "UIW_PROMPT."

- *text* is the text that is shown on the prompt.

UIW_PROMPT::UIW_PROMPT

Syntax

#include <ui_win.hpp>

UIW_PROMPT(int *left,* int *top,* ZIL_ICHAR *"text,*
    WOF_FLAGS *woFlags* = WOF_NO_FLAGS);
    or
UIW_PROMPT(int *left,* int *top,* int *width,* ZIL_ICHAR *"text,*
    WOF_FLAGS *woFlags* = WOF_NO_FLAGS);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These overloaded constructors create a new UIW_PROMPT object.

The first overloaded constructor creates a UIW_PROMPT.

- $left_{in}$ and top_{in} is the starting position of the prompt field within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values. The width and height of the prompt will be automatically determined.

- $text_{in}$ is the text that is shown on the prompt. A hotkey for the prompt may be specified by inserting the '&' character into the string before the desired hotkey character. For example, if the string *name" is to be displayed and 'N' is to be the hotkey, the string should be entered as "&Name." The '&' will not be displayed, but will cause the hotkey character to be drawn appropriately. If an '&' is required in the text that is displayed, place two '&' characters in the string (e.g., "A && B" will display as "A & B" and the prompt will not have a hotkey). In those environments that don't support hotkeys on objects (e.g., Macintosh, NEXTSTEP) the '&' character will not be displayed and will have no effect. If the end-user presses a prompt's hotkey, the field that was added to the window immediately after the prompt will be made current.

- $woFlags_{in}$ are flags (common to all window objects) that determine the general operation of the prompt object. The following flags (declared in **UI_WIN.HPP)** control the general presentation of, and interaction with, a UIW_PROMPT class object:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is 1/10th the size of a normal cell by default.

WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

WOF_NO_FLAGS—Does not associate any special window flags with the object. Setting this flag left-justifies the data. This flag should not be used in conjunction with any other WOF flags.

The second overloaded constructor creates a UIW_PROMPT.

- $left_{in}$ and top_{in} is the starting position of the prompt field within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the prompt. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value. The height of the prompt is determined automatically by the UIW_PROMPT object.

- $text_{in}$ is the text that is shown on the prompt. A hotkey for the prompt may be specified by inserting the '&' character into the string before the desired hotkey character. For example, if the string *name" is to be displayed and 'N' is to be the hotkey, the string should be entered as "&Name." The '&' will not be displayed, but will cause the hotkey character to be drawn appropriately. If an '&' is required in the text that is displayed, place two '&' characters in the string (e.g., "A && B" will display as "A & B" and the prompt will not have a hotkey). In those environments that don't support hotkeys on objects (e.g., Macintosh, NEXTSTEP) the '&' character will not be displayed and will have no effect. If the end-user presses a prompt's hotkey, the field that was added to the window immediately after the prompt will be made current.

- $woFlags_{in}$ are flags (common to all window objects) that determine the general operation of the prompt object. The following flags (declared in **UI_WIN.HPP)** control the general presentation of, and interaction with, a UIW_PROMPT class object:

WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

WOF_JUSTIFY_CENTER—Center-justifies the text within the displayed prompt.

WOF_JUSTIFY_RIGHT—Right-justifies the text within the displayed prompt.

WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is 1/10th the size of a normal cell by default.

WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

WOF_NO_FLAGS—Does not associate any special window flags with the object. Setting this flag left-justifies the data. This flag should not be used in conjunction with any other WOF flags.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Create a window and add it to the window manager.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 62, 10);
    *window
        + new UIW_BORDER
        + new UIW_TITLE(" Company Information ")
        + new UIW_PROMPT(2, 1, "&Name:")
        + new UIW_STRING(11, 1, 40, "", 256)
        + new UIW_PROMPT(2, 2, "&Address:")
        + new UIW_STRING(11, 2, 40, "", 256)
        + new UIW_STRING(11, 3, 40, "", 256)
        + new UIW_BUTTON(10, 6, 10, "&Save", BTF_NO_TOGGLE | BTF_AUTO_SIZE,
            WOF_BORDER | WOF_JUSTIFY_CENTER)
        + new UIW_BUTTON(25, 6, 10, "&Cancel", BTF_NO_TOGGLE | BTF_AUTO_SIZE,
            WOF_BORDER | WOF_JUSTIFY_CENTER)
        + new UIW_BUTTON(40, 6, 10, "&Help", BTF_NO_TOGGLE | BTF_AUTO_SIZE,
            WOF_BORDER | WOF_JUSTIFY_CENTER);
    *windowManager + window;
```

```
    // The prompt fields will automatically be destroyed when the window
    // is destroyed.
}
```

UIW_PROMPT::~UIW_PROMPT

Syntax

#include <ui_win.hpp>

virtual ~UIW_PROMPT(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_PROMPT object.

UIW_PROMPT::ClassName

Syntax

#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to _className.

UIW_PROMPT::DataGet

Syntax

#include <ui_win.hpp>

ZIL_ICHAR *DataGet(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function gets the text associated with the prompt object.

returnValue$_{out}$ is a pointer to the text associated with the prompt.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_PROMPT *prompt)

    ZIL_ICHAR *text = prompt->DataGet();
```

```
}
```

UIW_PROMPT::DataSet

Syntax

#include <ui_win.hpp>

void DataSet(ZIL_ICHAR **text);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function assigns new text to the prompt and redisplays the prompt. If no text is passed in (i.e., *text* is NULL), the prompt will be redrawn.

- $text_{in}$ is a pointer to the new text information to be displayed on the prompt. If the WOF_NO_ALLOCATE_DATA flag is set, *text* must be a string, allocated by the programmer, that is not destroyed until the UIW_PROMPT class object is destroyed. Otherwise, the information associated with this argument is copied by the UIW_-PROMPT class object.

Example

```
#include <ui_win.hpp>

ExampleFunction1(UIW_PROMPT *prompt)
{



    prompt->DataSet("&Close");
}
```

UIW_PROMPT::DrawItem

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual advanced function is used to draw the object. If the WOS_OWNERDRAW status is set for the object, this function will be called when drawing the prompt. This allows the programmer to derive a new class from UIW_PROMPT and handle the drawing of the prompt, if desired.

- $returnValue_{out}$ is a response based on the success of the function call. If the object is drawn the function returns a non-zero value. If the object is not drawn, 0 is returned.

- $event_{in}$ contains the run-time message that caused the object to be redrawn. *event.region* contains the region in need of updating. The following logical events may be sent to the **DrawItem( )** function:

  S_CURRENT, S_NON_CURRENT, S_DISPLAY_ACTIVE and **S_DISPLAY_INACTIVE**—Messages that cause the object to be redrawn.

  WM_DRAWITEM—A message that causes the object to be redrawn. This message is specific to Windows and OS/2.

  Expose—A message that causes the object to be redrawn. This message is specific to Motif.

- $ccode_{in}$ contains the logical interpretation of *event.*

UIW_PROMPT::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the prompt object. It is declared virtual so that any derived prompt class can override its default operation.

- *returnValue*$_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the prompt object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event( ):**

  S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

  S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

  S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to

the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

All other events are passed by **Event( )** to **UI_WINDOW_OBJECT::Event( )** for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event( )** function.

UIW_PROMPT::Information

Syntax

```
#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST request, void *data,
    ZIL_OBJECTID objectID = ID_DEFAULT);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- $returnValue_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- $request_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the prompt:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_COPY_TEXT—Copies the *text* associated with the object into a buffer provided by the programmer. If this request is sent, *data* must be the address of a buffer where the prompt's text will be copied. This buffer must be large enough to contain all of the characters associated with the prompt and the terminating NULL character.

  I_GET_TEXT—Returns a pointer to the text associated with the object. If this request is sent, *data* should be a doubly-indirected pointer to **ZIL_ICHAR.** This request does not copy the text into a new buffer.

  I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

  I_SET_TEXT—Sets the text associated with the object. This request will also redisplay the object with the new text, *data* should be a pointer to the new text.

  All other requests are passed by **Information**) to **UI_WINDOW_OBJECT::-Information()** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include<ui_win.hpp>

ExampleFunction()
{
        ZIL_ICHAR string[30];
        prompt-information(I_COPY_TEXT, &string),



        prompt1-Information (I_SET_TEXT, "First name:")
        prompt2->Information(I_SET_TEXT, "Last name:");
```

Storage Members

This section describes those class members that are used for storage purposes.

UIW_PROMPT::UIW_PROMPT

Syntax

#include <ui_win.hpp>

UIW_PROMPT(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));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_PROMPT by loading the object from a

data file. Typically, the programmer does not need to use this constructor. If a prompt is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_PROMPT::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_PROMPT from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1*.

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_PROMPT::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_PROMPT::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New()** function.

- *returnValue*$_{out}$ is a pointer to the object's **New( )** function.

UIW_PROMPT::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- $name_{in}$ is the name of the object to be stored.

- $file_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

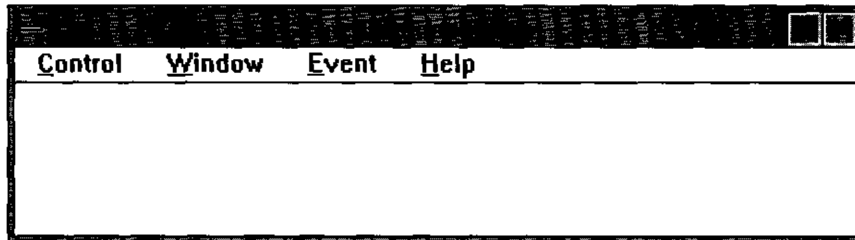
- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the

description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 17 - UIW_PULL_DOWN_ITEM

The UIW_PULL_DOWN_ITEM class is used to present the menu option categories in a window. Pull-down items are attached to a pull-down menu. Each pull-down item has an associated pop-up menu that presents the options available for that pull-down item. A pull-down item typically has a hotkey that allows it to be selected quickly from anywhere on the window. The figure below shows a graphical implementation of a pull-down menu containing several UIW_PULL_DOWN_ITEM objects (shown as "Control," "Window," "Event," and "Help"):



NOTE: On the Macintosh, a pull-down item cannot perform any action other than to display its pop-up menu when selected. Thus, while in other environments a pull-down item can place a message on the event queue or will call a user function, on the Macintosh it will do nothing.

The UIW_PULL_DOWN_ITEM class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_PULL_DOWN_ITEM : public UIW_BUTTON
{
public:
    static ZIL_ICHAR _className[];
    UIW_POP_UP_MENU menu;

    UIW_PULL_DOWN_ITEM(ZIL_ICHAR *text, WNF_FLAGS wnFlags = WNF_NO_FLAGS,
        ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION),
        EVENT_TYPE value = 0);
    UIW_PULL_DOWN_ITEM(ZIL_ICHAR *text, WNF_FLAGS wnFlags, UI_ITEM *item);
    virtual ~UIW_PULL_DOWN_ITEM(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_PULL_DOWN_ITEM(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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

    // List members.
    UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);
    UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT *object);
    UIW_PULL_DOWN_ITEM &operator+(UI_WINDOW_OBJECT *object);
    UIW_PULL_DOWN_ITEM ^operator-(UI_WINDOW_OBJECT *object);

protected:
    virtual EVENT_TYPE Drawltem(const UI_EVENT &event, EVENT_TYPE ccode);
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_PULL_DOWN_ITEM class, *_className* is "UIW_PULL_DOWN_ITEM."

- *menu* is the UIW_POP_UP_MENU that maintains the pop-up menu options that are displayed when the pull-down item is selected.

UIW_PULL_DOWN_ITEM::UIW_PULL_DOWN_ITEM

Syntax

#include <ui_win.hpp>

UIW_PULL_DOWN_ITEM(ZIL_ICHAR **text,*
    WNF_FLAGS *wnFlags* = WNF_NO_FLAGS,
    ZIL_USER_FUNCTION *userFunction* = ZIL_NULLF(ZIL_USER_FUNCTION),
    EVENT_TYPE *value* = 0);
    or
UIW_PULL_DOWN_ITEM(ZIL_ICHAR **text*, WNF_FLAGS *wnFlags,* UI_ITEM *"item);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These overloaded constructors create a new UIW_PULL_DOWN_ITEM class object.

The first constructor takes the following arguments:

- *text*$_{in}$ is the text that is shown in the pull-down item. A hotkey for the pull-down item may be specified by inserting the '&' character into the string before the desired hotkey character. For example, if the string "Exit" is to be displayed and 'x' is to be the hotkey, the string should be entered as "E&xit." The '&' will not be displayed, but will cause the hotkey character to be drawn appropriately. If an '&' is required in the text that is displayed, place two '&' characters in the string (e.g., "A && B" will display as "A & B" and the pull-down item will not have a hotkey). This string is copied by the UIW_PULL_DOWN_ITEM class unless the WOF_NO_ALLOCATE_DATA flag is set. If this flag is set, *text* must be space, allocated by the programmer, that is not deleted until the UIW_PULL_DOWN_ITEM object has been deleted.

- *wnFlags*$_{in}$ are flags that define the operation of the pull-down item's pop-up menu. The following flags (declared in **UI_WIN.HPP)** control the general presentation of the item's pop-up menu:

  WNF_AUTO_SORT—Causes the menu options to be sorted in alphabetical order.

  WNF_CONTINUE_SELECT—Allows the end-user to drag through the menu options with the mouse button pressed. If this flag is not set, the highlight on the menu options will not follow the dragging mouse. This flag should usually be set on a pop-up menu.

  WNF_NO_FLAGS—Does not associate any special flags with the pop-up menu. This flag should not be used in conjunction with any other WNF flags. This flag is set by default in the constructor.

WNF_NO_WRAP—Prevents the current option in the pop-up menu from wrapping between the top and bottom options when arrowing through the menu.

WNF_SELECT_MULTIPLE—Allows more than one option in the menu to become selected at the same time. If this flag is set, the pop-up menu will still close when a selection is made, but selecting another option later will not cause the previously selected item to be un-selected. This flag is typically used if the pop-up items in the menu have the MNIF_CHECK_MARK flag set.

- *userFunction*$_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. A user function should only be associated with a pull-down item if the item does not have a menu of options. As mentioned above, a pull-down item on the Macintosh will not perform any action other than to display an option menu. Thus, *userFunction* will be ignored on the Macintosh. The user function will be called by the library when:

  1—the user moves onto the field,

  2—the <ENTER> key is pressed while the pull-down item is current or the mouse is clicked on the object, or

  3—the user moves to a different field in the window or to a different window.

  Because the user function is called at these times, the programmer can do data validation or any other type of necessary operation. The definition of the *userFunction* is as follows:

  EVENT_TYPE FunctionName(UI_WINDOW_OBJECT **object,*
  UI_EVENT *&event,* EVENT_TYPE *ccode);*

  returnValue$_{out}$ indicates if an error has occurred. *returnValue* should be 0 if the no error occurred. Otherwise, the programmer should call the error system with an appropriate error message and return -1.

  object$_{in}$ is a pointer to the object for which the user function is being called. This argument must be typecast by the programmer if class-specific members need to be accessed.

  event$_{in}$ is the run-time message passed to the object.

$ccode_{in}$ is the logical or system code that caused the user function to be called. This code (declared in UI_WIN.HPP) will be one of the following constant values:

L_SELECT—The <ENTER> key was pressed while the field was current or the pull-down item was clicked on with the mouse.

S_CURRENT—The object just received focus because the user moved to it from another field or window.

S_NON_CURRENT—The object just lost focus because the user moved to another field or window.

- $value_{in}$ can be used as an identifier to distinguish between several pull-down items in a common user function. For example, the programmer could associate the value 0 with a "Save" menu item and a value of 1 with a "Save As" menu item. This allows the programmer to define one user function that can determine the action to take based on the pull-down item's value.

The second constructor creates a pull-down item with a pre-defined item array.

- $text_{in}$ is the text that is shown in the pull-down item. For a complete description of *text,* see the description of the first constructor.

- $wnFlags_{in}$ are flags that define the operation of the pull-down item's pop-up menu. For a complete description of *text,* see the description of the first constructor.

- $item_{in}$ is an array of UI_ITEM structures that are used to construct a set of pop-up items within the pull-down item's pop-up menu. For more information regarding the use of the UI_ITEM structure, see "Chapter 18—UI_ITEM" in *Programmer's Reference Volume 1.*

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Create a window with pull-down items.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_BORDER
        + new UIW_TITLE("Sample menus")
        + &(*new UIW_PULL_DOWN_MENU(1)
            + &(*new UIW_PULL_DOWN_ITEM("Item&1")
                + new UIW_POP_UP_ITEM("Option 1.1")
                + new UIW_POP_UP_ITEM("Option 1.2"))
```

```
            + &(*new UIW_PULL_DOWN_ITEM("Item&2")
                + new UIW_POP_UP_ITEM("Option 2.1"))
            + new UIW_PULL_DOWN_ITEM("Item&3"));
    *windowManager + window;



    // The pull-down items will automatically be destroyed when the window
    // is destroyed.
}

ExampleFunction2(UI_WINDOW_MANAGER  *windowManager)
{
    UI_ITEM item1[] =
    {
        { 11,       NULL,       "Option 1.1", MNIF_NO_FLAGS },
        { 12,       NULL,       "Option 1.2", MNIF_NO_FLAGS },
        { 0,        NULL,       NULL,            0 }
    };

    UI_ITEM item2[] =
    {
        { 21,       NULL,       "Option 2.1", MNIF_NO_FLAGS }.
        { 22,       NULL,       "Option 2.2", MNIF_NO_FLAGS },
        { 0,        NULL,       NULL,            0 }
    };

    // Create a window with pull-down items.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_BORDER
        + new UIW_TITLE("Sample menus")
        + &(*new UIW_PULL_DOWN_MENU(1)
            + new UIW_PULL_DOWN_ITEM("Item&1", WNF_NO_FLAGS, item1)
            + new UIW_PULL_DOWN_ITEM("Item&2", WNF_NO_FLAGS, item2)
            + new UIW_PULL_DOWN_ITEM("Item&3"));
    *windowManager + window;



    // The pull-down items will automatically be destroyed when the window
    // is destroyed.
}
```

UIW_PULL_DOWN_ITEM::~UIW_PULL_DOWN_ITEM

Syntax

#include <ui_win.hpp>

virtual ~UIW_PULL_DOWN_ITEM(void);

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_PULL_-DOWN_ITEM object. All objects attached to the pull-down item will also be destroyed.

UIW_PULL_DOWN_ITEM::Add
UIW_PULL_DOWN_ITEM::operator +

Syntax

#include <ui_win.hpp>

UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT **object);*
or
UIW_PULL_DOWN_ITEM &operator + (UI_WINDOW_OBJECT **object);*

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

These overloaded functions are used to add a new pop-up item to the UIW_PULL_-DOWN_ITEM object's menu.

The first overloaded function adds a new pop-up item to the UIW_PULL_DOWN_ITEM object's menu.

- $returnValue_{out}$ is a pointer to *object* if the addition was successful. Otherwise, *returnValue* is NULL.

• $object_{in}$ is a pointer to the pop-up item to be added to the menu.

The second overloaded operator adds a new pop-up item to the UIW_PULL_DOWN_-ITEM object's menu. This operator overload is equivalent to calling the Add() function except that it allows the chaining of pop-up item additions to the UIW_PULL_-DOWN_ITEM object.

• $returnValue_{out}$ is a pointer to the UIW_PULL_DOWN_ITEM object. This pointer is returned so that the operator may be used in a statement containing other operations.

• $object_{in}$ is a pointer to the new pop-up item that is to be added to the menu.

UIW_PULL_DOWN_ITEM::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

• DOS Text • DOS Graphics • Windows • OS/2
• Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual function returns the object's class name.

• $returnValue_{out}$ is a pointer to *_className.*

UIW_PULL_DOWN_ITEM::DrawItem

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual advanced function is used to draw the object. If the WOS_OWNERDRAW status is set for the object, this function will be called when drawing the pull-down item. This allows the programmer to derive a new class from UIW_PULL_DOWN_ITEM and handle the drawing of the pull-down item, if desired.

- $returnValue_{out}$ is a response based on the success of the function call. If the object is drawn, the function returns a non-zero value. If the object is not drawn, 0 is returned.

- $event_{in}$ contains the run-time message that caused the object to be redrawn. *event.region* contains the region in need of updating. The following logical events may be sent to the **DrawItem()** function:

  S_CURRENT, S_NON_CURRENT, S_DISPLAY_ACTIVE and S_DISPLAY_INACTIVE—Cause the object to be redrawn.

  WM_DRAWITEM—Causes the object to be redrawn. This message is specific to Windows and OS/2.

  Expose—Causes the object to be redrawn. This message is specific to Motif.

- $ccode_{in}$ contains the logical interpretation of *event.*

UIW_PU LL_DOWN_ITEM::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function processes run-time messages sent to the pull-down item object. It is declared virtual so that any derived pull-down item class can override its default operation.

- *returnValue*$_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the pull-down item object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event():**

  L_BEGIN_SELECT—Indicates that the end-user began the selection of the object by pressing the mouse button down while on the object.

  L_DOWN—Causes the pull-down item's menu to be displayed if it is not already displayed. This message is interpreted from a keyboard event.

  L_SELECT—Indicates that the object has been selected. The selection may be the result of a mouse click or a keyboard action.

  L_UP—Causes the pull-down item's menu to be displayed if it is not already displayed. This message is interpreted from a keyboard event.

S_ADD_OBJECT—Causes a new pop-up item to be added to the object's menu, *event.data* will point to the new pop-up item to be added.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_CURRENT—Causes the object to draw itself to appear current. This message is sent by the Window Manager to a window when it becomes current. The window, in turn, passes this message to the object on the window that is current.

S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager sends this message to a window when the window is subtracted from the Window Manager. The window, in turn, relays the message to all objects attached to it.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another field or window.

S_REDISPLAY—Causes the object to redraw.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

S_RESET_DISPLAY—Changes the display to a different resolution, *event.data* should point to the new display class to be used. If *event.data* is NULL, a text mode display will be created. This event is specific to DOS and must be placed on the event queue by the programmer. The library will never generate this event.

S_SUBTRACT_OBJECT—Causes a pop-up item to be subtracted from the object's menu, *event.data* will point to the pop-up item to be subtracted.

All other events are passed by **Event( )** to **UIW_BUTTON::Event( )** for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event( )** function.

UIW_PULL_DOWN_ITEM::Information

Syntax

```
#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST request, void *data,
    ZIL_OBJECTID objectID = ID_DEFAULT);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the pull-down item:

I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

I_GET_NUMBERID_OBJECT—Returns a pointer to an object whose *numberlD* matches the value in *data*, if one exists. This object does a depth-first search of the objects attached to it, looking for a match of the *numberlD.* If no object has a *numberlD* that matches *data,* NULL is returned. If this message is sent, *data* must be a pointer to a NUMBERID.

I_GET_STRINGID_OBJECT—Returns a pointer to an object whose *stringlD* matches the character string in *data,* if one exists. This object does a depth-first search of the objects attached to it, looking for a match of the *stringlD.* If no object has a *stringlD* that matches *data,* NULL is returned. If this message is sent, *data* must be a pointer to a string.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

All other requests are passed by **Information( )** to **UIW_BUTTON::Information( )** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

UIW_PULL_DOWN_ITEM::Subtract
UIW_PULL_DOWN_ITEM::operator -

Syntax

#include <ui_gen.hpp>

UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT **object);*
    or
UIW_PULL_DOWN_ITEM &operator - (UI_WINDOW_OBJECT **object);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These functions remove an element from the UIW_PULL_DOWN_ITEM object's menu.

The first function removes a pop-up item from the UIW_PULL_DOWN_ITEM object's menu but does not call the destructor associated with the object. The programmer is responsible for deletion of each object explicitly subtracted from a list.

- $returnValue_{out}$ is a pointer to the next pop-up item in the menu. This value is NULL if there are no more pop-up items after the subtracted pop-up item.

- $element_{in}$ is a pointer to the pop-up item to be subtracted from the menu.

The second overloaded operator removes a pop-up item from the UIW_PULL_DOWN_-ITEM object's menu but does not call the destructor associated with the object. This operator overload is equivalent to calling the **Subtract()** function, except that it allows the chaining of pop-up item removals from the UIW_PULL_DOWN_ITEM object's menu.

- $returnValue_{out}$ is a pointer to the UIW_PULL_DOWN_ITEM object. This pointer is returned so that the operator may be used in a statement containing other operations.

- $object_{in}$ is a pointer to the pop-up item that is to be subtracted from the menu.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_PULL_DOWN_ITEM::UIW_PULL_DOWN_ITEM

Syntax

```
#include <ui_win.hpp>

UIW_PULL_DOWN_ITEM(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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_PULL_DOWN_ITEM by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a pull-down item is stored in a data file it is usually stored as part of a UIW_-WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter

69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW PULL_DOWN ITEM::Load

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This advanced function is used to load a UIW_PULL_DOWN_ITEM from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW PULL DOWN_ITEM::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1*.

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_PULL_DOWN_ITEM::NewFunction

Syntax

#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New( )** function.

- *returnValue*$_{out}$ is a pointer to the object's **New( )** function.

UIW_PULL_DOWN_ITEM::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- $name_{in}$ is the name of the object to be stored.

- $file_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

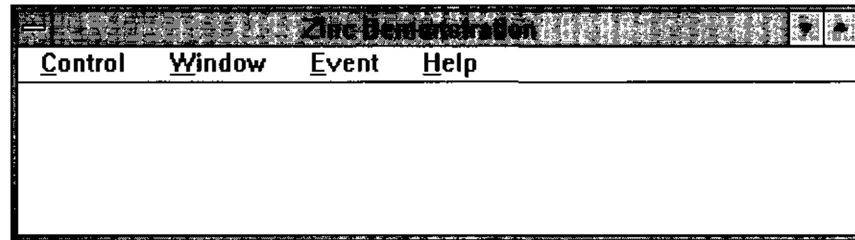
- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 18 - UIW_PULL_DOWN_MENU

The UIW_PULL_DOWN_MENU class object is used as a controlling structure for menu items. The pull-down menu appears as a line of options across the top of the window, immediately below the title bar. If more options are added to the menu than can fit on a single line, or if the window is sized narrower so that the options will not fit on one line, the menu will expand and the options will wrap so that they are displayed on multiple lines. The figure below shows a graphical implementation of a UIW_PULL_-DOWN_MENU class object with four pull-down items (shown as "Control," "Window," "Event," and "Help"):



NOTE: Microsoft Windows does not allow pull-down menus on child windows. This applies whether the child window is an MDI child or simply a child window.

On the Macintosh, only one pull-down menu per application is displayed at any given time. If a window other than the one with which the pull-down menu is associated becomes current, and the new current window does not have a pull-down menu, the selections on the pull-down menu will become non-selectable. If the original window becomes current again, these selections will become selectable.

The UIW_PULL_DOWN_MENU class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT__CLASS UIW_PULL_DOWN_MENU : public UIW_WINDOW
{
public:
    static ZIL_ICHAR _className[];

    UIW_PULL_DOWN_MENU(int indentation = 0,
        WOF_FLAGS woFlags = WOF_BORDER | WOF_NON_FIELD_REGION |
            WOF_SUPPORT_OBJECT,
        WOAF_FLAGS woAdvancedFlags = WOAF_NO_FLAGS);
    UIW_PULL_DOWN_MENU(int indentation, UI_ITEM *item);
    virtual ~UIW_PULL_DOWN_MENU(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);
```

```
#if defined(ZIL_MACINTOSH)
    UIW_POP_UP_ITEM *ItemDepthSearch(long mSelect);
#endif

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_PULL_DOWN_MENU(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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

    // List members.
#if defined(ZIL_MACINTOSH)
    virtual UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);
    UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT *object);
    UIW_PULL_DOWN_MENU &operator+(UI_WINDOW_OBJECT *object);
    UIW_PULL_DOWN_MENU &operator-(UI_WINDOW_OBJECT *object);
#endif

protected:
    int indentation;
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_PULL_DOWN_MENU class, *_className* is "UIW_PULL_DOWN_MENU."

- *indentation* is the number of cells over from the left edge of the menu where the first menu item should be displayed. The indented space is shown as blank space in the menu.

UIW_PULL_DOWN_MENU::UIW_PULL_DOWN_MENU

Syntax

```
#include <ui_win.hpp>

UIW_PULL_DOWN_MENU(int indentation = 0,
    WOF_FLAGS woFlags = WOF_BORDER | WOF_NON_FIELD_REGION |
        WOF_SUPPORT_OBJECT,
    WOAF_FLAGS woAdvancedFlags = WOAF_NO_FLAGS);
    or
UIW_PULL_DOWN_MENU(int indentation, UI_ITEM *item);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These overloaded constructors create a new UIW_PULL_DOWN_MENU object.

The first constructor creates a UIW_PULL_DOWN_MENU object.

- $indentation_{in}$ is the number of cells over from the left edge of the menu where the first menu item should be displayed. The indented space is shown as blank space in the menu. Subsequent menu items will be automatically positioned next to the previous menu item.

- $woFlags_{in}$ are flags (common to all window objects) that determine the general operation of the pull-down menu object. The following flags (declared in **UI_WIN.HPP)** control the general presentation of, and interaction with, a UIW_-PULL_DOWN_MENU class object:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support

Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to use the remaining available space in its parent object. This flag should always be set on a pull-down menu. It is set by default in the constructor.

WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor select any menu items. Typically, the object will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

WOF_SUPPORT_OBJECT—Causes the object to be placed in the parent object's support list. The support list is reserved for objects that are not displayed as part of the user region of the window. This flag should be set for a pull-down menu. This flag is set by default in the constructor.

- $woAdvancedFlags_{in}$ are flags (common to all window objects) that determine the advanced operation of the pull-down menu object. The following flags (declared in **UI_WIN.HPP)** control the advanced operation of a pull-down menu object:

  WOAF_NO_FLAGS—Does not associate any special advanced flags with the window object. This flag should not be used in conjunction with any other WOAF flags.

  WOAF_NORMAL_HOT_KEYS—Allows the end-user to select an option using its hotkey by pressing the hotkey by itself, without the <Alt> key otherwise required for selecting with a hotkey.

The second constructor creates a pull-down menu using a pre-defined item array. These items are used to create UIW_PULL_DOWN_ITEM objects. **NOTE:** This constructor generally should not be used in Macintosh applications as the Macintosh requires pull-down items to have sub-menus that perform an action.

- $indentation_{in}$ is the number of cells over from the left edge of the menu where the first menu item should be displayed. The indented space is shown as blank space in the menu. Subsequent menu items will be automatically positioned next to the previous menu item.

• $item_{in}$ is an array of UI_ITEM structures that are used to construct a set of pull-down items within the pull-down menu. For more information regarding the use of the UI_ITEM structure, see "Chapter 18—UI_ITEM" in *Programmer's Reference Volume 1.*

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER  *windowManager)
{
    UI ITEM items1

        { 1        MenuFunction,       "Option 1",     WNF_NO_FLAGS },
        { 2        MenuFunction,       "Option 2",     WNF_NO_FLAGS }
        { 3        MenuFunction,       "Option 3",     WNF_NO_FLAGS }
        { 4        MenuFunction,       "Option 4",     WNF_NO_FLAGS }
        { 0        NULL,               NULL,           WNF_NO_FLAGS }
    };
    // Create a window with pull-down items.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10)
    *window
        + new UIW_BORDER
        + new UIW_TITLE(" Sample menus ")
        + new UIW_PULL_DOWN_MENU(1, items);

    *windowManager + window;



    // The pull-down menu and pull-down items will automatically be destroyed
    // when the window is destroyed.
```

UIW PULL DOWN MENU::~UIW PULL DOWN MENU

Syntax

#include <ui_win.hpp>

virtual ~UIW_PULL_DOWN_MENU(void);

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_PULL_-DOWN_MENU object. All objects attached to the pull-down menu will also be destroyed.

UIW_PULL_DOWN_MENU::Add

Syntax

```
#include <ui_win.hpp>

UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);
    or
UIW_PULL_DOWN_MENU &operator + (UI_WINDOW_OBJECT *object);
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function adds a new pull-down item to the UIW_PULL_DOWN_MENU object.

- *returnValue*$_{out}$ is a pointer to *object* if the addition was successful. Otherwise, *returnValue* is NULL.

- *object*$_{in}$ is a pointer to the pull-down item to be added to the pull-down menu.

The second overloaded operator adds a pull-down item to the UIW_PULL_DOWN_-MENU. This operator overload is equivalent to calling the Add() function, except that it allows the chaining of pull-down item additions to the UIW_PULL_DOWN_MENU.

returnValue$_{out}$ is a pointer to the UIW_PULL_DOWN_MENU object. This pointer is returned so that the operator may be used in a statement containing other operations.

- *object*$_{in}$ is a pointer to the pull-down item that is to be added to the menu.

UIW_PULL_DOWN_MENU::ClassName

Syntax

#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to *_className.*

UIW_PULL_DOWN_MENU::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function processes run-time messages sent to the pull-down menu object. It is declared virtual so that any derived pull-down menu class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the pull-down menu object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by Event():

  L_BEGIN_SELECT—Indicates that the end-user began the selection of the object by pressing the mouse button down while on the object.

  S_ADD_OBJECT—Causes a new pull-down item to be added to the pull-down menu, *event.data* will point to the new pull-down item to be added.

  S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

  S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

  S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager sends this message to a window when the window is subtracted from the Window Manager. The window, in turn, relays the message to all objects attached to it.

  S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

  S_REDISPLAY—Causes the object to redraw.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

S_SUBTRACT_OBJECT—Causes a pull-down item to be subtracted from the pull-down menu, *event.data* will point to the pull-down item to be subtracted.

All other events are passed by **Event( )** to **UIW_WINDOW::Event( )** for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event( )** function.

UIW_PULL_DOWN_MENU::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request*, void **data,*
ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the pull-down menu:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

  All other requests are passed by **Information()** to **UIW_WINDOW::Information()** for processing.

- *data*$_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- *objectID*$_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID*, the object will attempt to interpret the request with the *objectID* of the actual object type.

UIW_PULL_DOWN_MENU::ItemDepthSearch

Syntax

```
#include <ui_win.hpp>

UIW_POP_UP_ITEM *ItemDepthSearch(long mSelect);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function returns a pointer to the pop-up item identified by *mSelect.* This function is available for the Macintosh only.

- $returnValue_{out}$ is a pointer to the pop-up item requested. If no pop-up item matched *mSelect*, NULL is returned.

- $mSelect_{in}$ is an identifier for the pop-up item requested.

UIW_PULL_DOWN_MENU::Subtract
UIW_PULL_DOWN_MENU::operator -

Syntax

#include <ui_gen.hpp>

UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT **object);*
or
UIW_PULL_DOWN_MENU &operator - (UI_WINDOW_OBJECT **object);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These functions remove pull-down item from the UIW_PULL_DOWN_MENU.

The first function removes a pull-down item from the UIW_PULL_DOWN_MENU but does not call the destructor associated with the object. The programmer is responsible for deletion of each object explicitly subtracted from a list.

- $returnValue_{out}$ is a pointer to the next pull-down item in the menu. This value is NULL if there are no more pull-down items after the subtracted pull-down item.

- $element_{in}$ is a pointer to the pull-down item to be subtracted from the menu.

The second overloaded operator removes a pull-down item from the UIW_PULL_-DOWN_MENU but does not call the destructor associated with the object. This operator overload is equivalent to calling the **Subtract^)** function, except that it allows the chaining of pull-down item removals from the UIW_PULL_DOWN_MENU.

- $returnValue_{out}$ is a pointer to the UIW_PULL_DOWN_MENU object. This pointer is returned so that the operator may be used in a statement containing other operations.

- $object_{in}$ is a pointer to the pull-down item that is to be subtracted from the menu.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_PULL_DOWN_MENU::UIW_PULL_DOWN_MENU

Syntax

#include <ui_win.hpp>

UIW_PULL_DOWN_MENU(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));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_PULL_DOWN_MENU by loading the object from a data file. Typically, the programmer does not need to use this constructor.

If a pull-down menu object is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1*.

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1*.

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_PULL_DOWN_MENU::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_PULL_DOWN_MENU from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_PULL_DOWN_MENU::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECTJREAD_ONLY" of *Programmer's Reference Volume 1.*

objectTable$_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

userTable$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_PULL_DOWN_MENU::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New( )** function.

- *returnValue*$_{out}$ is a pointer to the object's **New( )** function.

UIW_PULL_DOWN_MENU::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- $name_{in}$ is the name of the object to be stored.

- $file_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

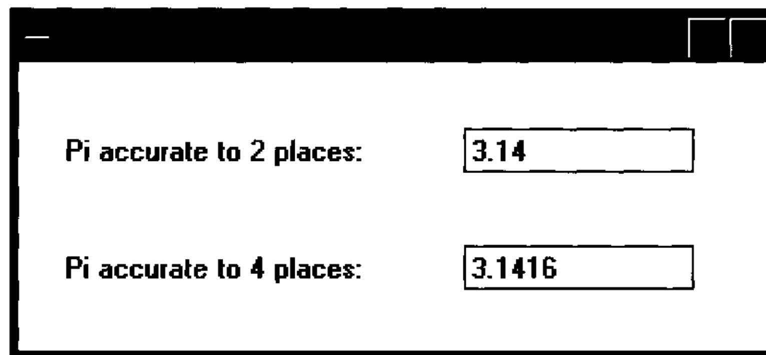
- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the

description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 19 - UIW_REAL

The UIW_REAL class is used to display floating-point information and to collect information, in floating-point format, from the end-user. The UIW_REAL class will display **double** values using decimal notation. If larger values are required or if any formatting is necessary (e.g., currency symbols) the UIW_BIGNUM object should be used. The figure below shows the graphical implementation of a window with UIW_REAL class objects:



The UIW_REAL class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_REAL : public UIW_STRING
{
public:
    static ZIL_ICHAR _className[];
    static int defaultInitialized;
    NMF_FLAGS nmFlags;

    UIW_REAL(int left, int top, int width, double *value,
        const ZIL_ICHAR *range = ZIL_NULLP(ZIL_ICHAR),
        NMF_FLAGS nmFlags = NMF_NO_FLAGS,
        WOF_FLAGS woFlags = WOF_BORDER | WOF_AUTO_CLEAR,
        ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION));
    virtual ~UIW_REAL(void);
    virtual ZIL_ICHAR *ClassName(void);
    double DataGet(void);
    void DataSet(double *value);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);
    virtual int Validate(int processError = TRUE);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_REAL(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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

    void SetLanguage(const ZIL_ICHAR *languageName);

protected:
    double *number;
    ZIL_ICHAR *range;
    const ZIL_LANGUAGE *myLanguage;
    static void Format(ZIL_ICHAR *text, double number, NMF_FLAGS flags);
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_REAL class, *_className* is "UIW_REAL."

- *defaultlnitialized* indicates if the default language strings for this object have been set up. The default strings are located in the file LANG_DEF.CPP. If *defaultlnitialized* is TRUE, the strings have been set up. Otherwise they have not been.

- *nmFlags* are flags that define the operation of the UIW_REAL class. A full description of the number flags is given in the UIW_REAL constructor.

- *number* is used to store the double value for UIW_REAL. If the WOF_NO_-ALLOCATE_DATA flag is set, *number* will simply point to the value that was passed in the constructor.

- *range* is a string that specifies the range(s) of acceptable values, *range* is a copy of the range that is passed to the constructor.

- *myLanguage* is the ZIL_LANGUAGE object that contains the string translations for this object.

UIW_REAL::UIW_REAL

Syntax

```
#include <ui_win.hpp>

UIW_REAL(int left, int top, int width, double *value,
    const ZIL_ICHAR *range = ZIL_NULLP(ZIL_ICHAR),
    NMF_FLAGS nmFlags = NMF_NO_FLAGS,
    WOF_FLAGS woFlags = WOF_BORDER | WOF_AUTO_CLEAR,
    ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_REAL class object.

- $left_{in}$ and top_{in} is the starting position of the real field within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the real field. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value. The height of the real field is determined automatically by the UIW_REAL object.

- $value_{in}$ is a pointer to the default numeric value. This value is copied into a buffer allocated by the UIW_REAL object unless the WOF_NO_ALLOCATE_DATA flag is set, in which case *value* is used.

- $range_{in}$ is a string that specifies the valid numeric ranges. A range consists of a minimum value, a maximum value, and the values in between. For example, if a range of "10.00..100.00" is specified, the UIW_REAL class object will only accept those numeric values that fall between 10.00 and 100.00, inclusive. Open-ended ranges can be specified by leaving the minimum or maximum value off. For

example, a range of "50.00.." will allow all values that are 50.00 or greater. Multiple, disjoint ranges can be specified by separating the individual ranges with a slash (i.e. '/'). For example, "10.00..19.90/100.00.." will accept all values from 10.00 to 19.90 and values of 100.00 or greater. If *range* is NULL, any number within the absolute range is accepted. This string is copied by the UIW_REAL class object to the *range* member variable.

- $nmFlags_{in}$ describes how the real object should display and interpret the numeric information. The following flags (declared in UI_GEN.HPP) control the general presentation of a UIW_REAL class object:

  NMF_DIGITS*(number)*—Sets the number of decimal places to *number* decimal places.

  NMF_NO_FLAGS—Does not associate any special flags with the number object. This flag should not be used in conjunction with any other NMF flag. This is the default argument in the constructor.

  NMF_SCIENTIFIC—Causes the number to be formatted with scientific notation.

- $woFlags_{in}$ are flags (common to all window objects) that determine the general operation of the real object. The following flags (declared in UI_WIN.HPP) affect the operation of a UIW_REAL class object:

  WOF_AUTO_CLEAR—Automatically marks the entire buffer if the end-user tabs to the field from another object. If the user then enters data (without first having pressed any movement or editing keys) the entire field will be replaced. This flag is set by default in the constructor.

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOF_INVALID—Sets the initial status of the field to be "invalid." Invalid entries fit in the absolute range determined by the object type but do not fulfill all the requirements specified by the program. For example, a real may initially be set to 20.00, but the final number, edited by the end-user, must be in the range "1.00.. 10.00." The initial number in this example fits the absolute range

requirements of a UIW_REAL class object but does not fit into the specified range. By denoting the field as invalid, the user is forced to enter an acceptable value.

WOF_JUSTIFY_CENTER—Center-justifies the data within the displayed field.

WOF_JUSTIFY_RIGHT—Right-justifies the data within the displayed field.

WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is l/10th the size of a normal cell by default.

WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

WOF_NO_FLAGS—Does not associate any special window flags with the object. Setting this flag left-justifies the data. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor edit the real information. Typically, the field will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

WOF_SUPPORT_OBJECT—Causes the object to be placed in the parent object's support list. The support list is reserved for objects that are not displayed as part of the user region of the window. Care should be used when setting this flag on an object that does not use it by default as undesirable effects may occur. This flag generally should not be used by the programmer.

WOF_UNANSWERED—Sets the initial status of the field to be "unanswered." An unanswered field is displayed as an empty field.

WOF_VIEW_ONLY—Prevents the object from being edited. However, the object may become current and the user may scroll through the data, mark it, and copy it.

- *userFunction*$_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. The user function will be called by the library when:

  1—the user moves onto the field,

  2—the <ENTER> key is pressed while the field is current or, if the field is in a list, the mouse is clicked on it, or

  3—the user moves to a different field in the window or to a different window.

  Because the user function is called at these times, the programmer can do data validation or any other type of necessary operation. The definition of the *userFunction* is as follows:

  EVENT_TYPE FunctionName(UI_WINDOW_OBJECT **object,*
  UI_EVENT *&event,* EVENT_TYPE *ccode);*

  returnValue$_{out}$ indicates if an error has occurred. *returnValue* should be 0 if the no error occurred. Otherwise, the programmer should call the error system with an appropriate error message and return -1.

  object$_{in}$ is a pointer to the object for which the user function is being called. This argument must be typecast by the programmer if class-specific members need to be accessed.

  event$_{in}$ is the run-time message passed to the object.

  ccode$_{in}$ is the logical or system code that caused the user function to be called. This code (declared in UI_WIN.HPP) will be one of the following constant values:

  L_SELECT—The <ENTER> key was pressed while the field was current, or, if the field is in a list, the mouse was clicked on the field.

  S_CURRENT—The object just received focus because the user moved to it from another field or window. This code is sent before any editing operations are permitted.

  S_NON_CURRENT—The object just lost focus because the user moved to another field or window.

NOTE: If a user function is associated with the object, **Validate()** must be called explicitly from within *userFunction* if range checking is desired.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Create a window and add it to the window manager,
    double value = 0.0;
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIWJBORDER
        + new UIW_TITLE(" Sample numbers ")
        + new UIW_PROMPT(2, 1, "Standard:")
        + new UIW_REAL(12, 1, 20, &value, "0..10000")
        + new UIW_PROMPT(2, 2, "Currency:");
    *windowManager + window;



    // The real number fields are automatically destroyed when the window
    // is destroyed.
}
```

UIW REAL::~UIW REAL

Syntax

#include <ui_win.hpp>

virtual ~UIW_REAL(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_REAL object.

UIW_REAL::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- $returnValue_{out}$ is a pointer to *_className.*

UIW_REAL::DataGet

Syntax

```
#include <ui_win.hpp>

double DataGet(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function gets the current numeric information associated with the UIW_REAL class

object.

- *returnValue*$_{out}$ is the **double** value.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_REAL *realObject)
{
    double value = realObject->DataGet();


}
```

UIW_REAL::DataSet

Syntax

#include <ui_win.hpp>

void DataSet(double *"value);*

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function assigns a new value to the UIW_REAL object and redisplays the field. If no value is passed in (i.e., *value* is NULL), the field will be redrawn.

- *value*$_{in}$ is a pointer to the new value. If the WOF_NO_ALLOCATE_DATA flag is set, this argument must be a **double,** allocated by the programmer, that is not destroyed until the UIW_REAL class object is destroyed. Otherwise, the information associated with this argument is copied by the UIW_REAL class object. If this argument is NULL, no numeric information is changed, but the number field is redisplayed.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_REAL *number)
{
    double amount = 100.0;
    number->DataSet(&amount);
}
```

UIW_REAL::Event

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the real object. It is declared virtual so that any derived real class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the real object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by Event():

  L_SELECT—Indicates that the object has been selected. The selection may be the result of a mouse click or a keyboard action.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another field or window.

All other events are passed by **Event()** to **UIW_STRING::Event()** for processing.

UIW_REAL::Format

Syntax

#include <ui_win.hpp>

static void Format(ZIL_ICHAR **text,* double *number,* NMF_FLAGS *flags);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function places the string representation of *number* in *text.* The text is formatted according to the flags passed in *flags.*

- *text*$_{in}$ is a pointer to a buffer where the string representation of the number is placed. This string must be long enough to contain the entire string including the NULL terminator.

- *number*$_{in}$ is the value whose string representation will be converted.

- *flags*$_{in}$ specifies the format for the text. The following flags (defined in **UI_-WIN.HPP)** are valid:

NMF_DIGITS(number)—Sets the number of decimal places to *number* decimal places.

NMF_NO_FLAGS—Does not associate any special flags with the number object. This flag should not be used in conjunction with any other NMF flag.

NMF_SCIENTIFIC—Causes the number to be formatted with scientific notation.

UIW REAL::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request,* void **data,*
    ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc objects and programmer functions to get or modify specified information about an object.

- $returnValue_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- $request_{in}$ is a request to get or set information associated with the object. The following requests (defined in UI_WIN.HPP) are recognized by the real:

    I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags,

particularly if the new flag settings will change the visual appearance of the object.

I_DECREMENT_VALUE—Decrements the real's value. If this message is sent, *data* must be a pointer to an **ZIL_INT32** (only integral values are supported for this request). The real object's value will be decremented by the value of *data.* The real will not be modified if the new value is not within the specified range.

I_GET_VALUE—Returns the *value* associated with the real. If this message is sent, *data* must be a pointer to a variable of type **double** where the real's value will be copied.

I_INCREMENT_VALUE—Increments the real's value. If this message is sent, *data* must be a pointer to an **ZIL_INT32** (only integral values are supported for this request). The real object's value will be incremented by the value of *data.* The real will not be modified if the new value is not within the specified range.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_VALUE—Sets the *value* associated with the real. If this message is sent, *data* must be a pointer to a variable of type **double** that contains the real's new value.

All other requests are passed by **Information**() to **UIW_STRING::Information()** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information**() function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

UIW_REAL::SetLanguage

Syntax

```
#include <ui_win.hpp>

void SetLanguage(const ZIL_ICHAR *languageName);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function sets the language to be used by the object. The string translations for the object will be loaded and the object's *myLanguage* member will be updated to point to the new ZIL_LANGUAGE object. By default, the object uses the language identified in the **LANG_DEF.CPP** file, which compiles into the library. (If a different default language is desired, simply copy a **LANG_<ISO>.CPP** file from the OpenZinc\SOURCE\-INTL directory to the \OpenZinc\SOURCE directory, and rename it to **LANG_DEF.CPP** before compiling the library.) The language translations are loaded from the **I18N.DAT** file, so it must be shipped with your application.

- $languageName_{in}$ is the two-letter ISO language code identifying which language the object should use.

UIW_REAL::Validate

Syntax

```
#include <ui_win.hpp>

virtual int Validate(int processError = TRUE);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function is used to validate objects. When an object receives the S_CURRENT or S_NON_CURRENT messages, it calls **Validate()** to check if the value entered is valid. However, if a user function is associated with the object, **Validate()** must be called explicitly from the user function if range checking is desired. The value is invalid if it is not within the absolute range of the object or if it is not within a range specified by the *range* member variable.

- $returnValue_{out}$ indicates the result of the validation. The possible values for *returnValue* are:

  NMI_GREATER_THAN_RANGE—The number entered was greater than the maximum value of a negatively open-ended range.

  NMI_INVALID—The number was entered in an incorrect format.

  NMI_LESS_THAN_RANGE—The number entered was less than the minimum value of a positively open-ended range.

  NMI_OK—The number was entered in a correct format and within the valid ranges.

  NMI_OUT_OF_RANGE—The number was not within the valid range for numbers or was not within the specified *range.*

- $processError_{in}$ determines whether **Validate()** should call **UI_ERROR_SYSTEM::-ReportError()** if an error occurs. If *processError* is TRUE, **ReportError( )** is called. Otherwise, the error system is not called.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_REAL::UIW_REAL

Syntax

```
#include <ui_win.hpp>

UIW_REAL(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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_REAL by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a real is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL,

the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_REAL::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced function is used to load a UIWJREAL from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the

programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_REAL::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW REAL::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New( )** function.

returnValue$_{out}$ is a pointer to the object's **New( )** function.

UIW REAL::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI*W_W*INDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 20 - UIW_SCROLL_BAR

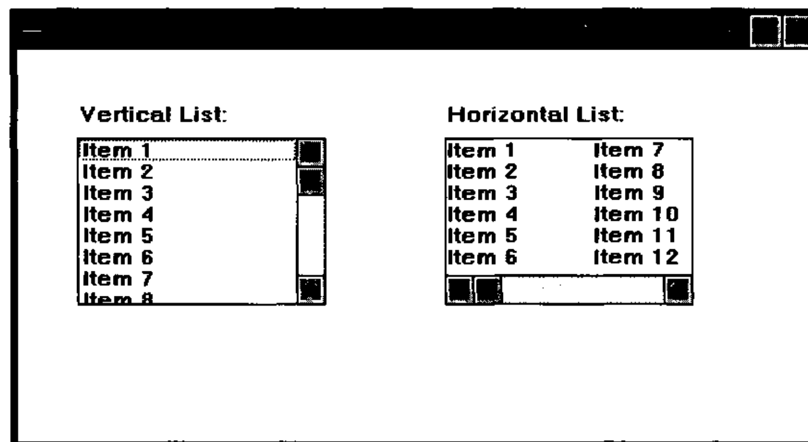
The UIW_SCROLL_BAR class is used to create a scroll bar or a slider. There are two differences between a scroll bar and a slider. The first difference is their appearance. In most environments they look a little different. The scroll bar track is usually as wide as the scroll bar whereas the slider track is typically somewhat thinner.

The second difference is how a vertical scroll bar scrolls as opposed to a vertical slider. As the scroll position on a vertical scroll bar increases, its thumb button moves from the top of the scroll bar toward the bottom. On a vertical slider, however, the thumb button moves from the bottom of the slider toward the top as the position increases.

A scroll bar is typically used to scroll another object, such as a list.

A slider is typically used to visually indicate the current value relative to the range of possible values and to allow the setting of the value.

The figure below shows a graphical implementation of a list with a UIW_SCROLL_BAR class object:



The UIW_SCROLL_BAR class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_SCROLL_BAR : public UIW_WINDOW
{
public:
    static ZIL_ICHAR _className[];
    SBF_FLAGS sbFlags;
```

```
    UIW_SCROLL_BAR(int left, int top, int width, int height,
        SBF_FLAGS sbFlags = SBF_VERTICAL,
        WOF_FLAGS   woFlags   =   WOF_BORDER   |   WOF_SUPPORT_OBJECT  |
            WOF_NON_FIELD_REGION);
    UIW_SCROLL_BAR(int left, int top, int width, int height,
        UI_SCROLL_INFORMATION *scroll, SBF_FLAGS sbFlags = SBF_VERTICAL,
        WOF_FLAGS woFlags = WOF_BORDER,
        WOAF_FLAGS woAdvancedFlags = WOAF_NON_CURRENT,
        ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION));
    virtual ~UIW_SCROLL_BAR(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Drawltem(const UI_EVENT &event, EVENT_TYPE ccode);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_SCROLL_BAR(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
#if defined (ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

protected:
    UI_SCROLL_INFORMATION scroll;
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_SCROLL_BAR class, *_className* is "UIW_SCROLL_BAR."

- *sbFlags* are flags that define the operation of the UIW_SCROLL_BAR class. A full description of the scroll bar flags is given in the UIW_SCROLL_BAR constructor.

- *scroll* contains the current scroll bar position information.

UIW_SCROLL_BAR::UIW_SCROLL_BAR

Syntax

```
#include <ui_win.hpp>

UIW_SCROLL_BAR(int left, int top, int width, int height,
    SBF_FLAGS sbFlags = SBF_VERTICAL,
    WOF_FLAGS woFlags = WOF_BORDER | WOF_SUPPORT_OBJECT |
        WOF_NON_FIELD_REGION);
    or
UIW_SCROLL_BAR(int left, int top, int width, int height,
    UI_SCROLL_INFORMATION *scroll, SBF_FLAGS sbFlags = SBF_VERTICAL,
    WOF_FLAGS woFlags = WOF_BORDER,
    WOAF_FLAGS woAdvancedFlags = WOAF_NON_CURRENT,
    ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These overloaded constructors create a new UIW_SCROLL_BAR class object.

The first constructor creates a scroll bar. A scroll bar is added to the object it is to control. The object must know how to respond to scrolling messages (e.g., S_VSCROLL and S_HSCROLL).

- *left*$_{in}$ and *top*$_{in}$ is the starting position of the scroll bar within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- *width*$_{in}$ is the width of the scroll bar. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- *height*$_{in}$ is the height of the scroll bar. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- *sbFlags*$_{in}$ are flags that define the operation of the UIW_SCROLL_BAR class. The following flags (declared in **UI_WIN.HPP)** control the general presentation of a UIW_SCROLL_BAR class object:

  SBF_CORNER—Creates a corner scroll bar object. A corner scroll bar fills the area between the horizontal and vertical scroll bars. This flag has no effect under Windows or OS/2, since these environments automatically draw the corner area.

  SBF_HORIZONTAL—Creates a horizontal scroll bar object.

  SBF_NO_FLAGS—Does not associate any special flags with the UIW_-SCROLL_BAR class object. In general, this flag should not be used since the SBF_FLAGS determine the type of scroll bar to create.

  SBF_VERTICAL—Creates a vertical scroll bar object.

- *woFlags*$_{in}$ are flags (common to all window objects) that determine the general operation of the scroll bar object. The following flags (declared in **UI_WIN.HPP)** control the general presentation of, and interaction with, a UIW_SCROLL_BAR class object:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is 1/10th the size of a normal cell by default.

  WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object. A vertical scroll bar will be placed along the right edge of its parent, while a horizontal scroll bar will be placed along the bottom edge of its parent. This flag is set by default in the constructor.

WOF_SUPPORT_OBJECT—Causes the object to be placed in the parent object's support list. The support list is reserved for objects that are not displayed as part of the user region of the window. If the WOF_SUPPORT_-OBJECT flag is set, the scroll bar will send scroll events to the object it controls. If this flag is not set, it will attempt to call its user function. For a scroll bar, this flag must be set. This flag is set by default in the constructor.

NOTE: Scroll bars are added directly to the object which they are to control.

The second constructor creates a slider control.

- $left_{in}$ and top_{in} is the starting position of the slider within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the slider. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $height_{in}$ is the height of the slider. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $scroll_{in}$ is the scroll information that defines the range and update values for the slider. For more information about the use of *scroll*, see "Chapter 39—UI_-SCROLLJNFORMATION" of *Programmer's Reference Volume 1*.

  scroll.current specifies the initial value of the slider.

  scroll.minimum specifies the minimum value of the slider range.

  scroll.maximum specifies the maximum value of the slider range.

  scroll.showing specifies the increment or decrement value for page up or page down movement.

scroll delta specifies the increment or decrement value for single interval up or down movement.

- *$sbFlags_{in}$* are flags that define the operation of the UIW_SCROLL_BAR class. The following flags (declared in **UI_WIN.HPP)** control the general presentation of a UIW_SCROLL_BAR class object:

  SBF_HORIZONTAL—Creates a horizontal slider object.

  SBF_NO_FLAGS—Does not associate any special flags with the UIW_-SCROLL_BAR class object. In general, this flag should not be used since the SBF_FLAGS determine the type of slider to create.

  SBF_SLIDER—Causes the object to appear as a slider. In most environments the slider looks different than a scroll bar. When set with the SBF_VERTICAL flag, the SBF_SLIDER flag also allows the vertical slider to update properly. A vertical slider's value increases as the thumb button moves toward the top of the shaft, whereas a scroll bar's value decreases as the thumb button moves up.

  SBF_VERTICAL—Creates a vertical slider object.

- *$woFlags_{in}$* are flags (common to all window objects) that determine the general operation of the slider object. The following flags (declared in **UI_WIN.HPP)** control the general presentation of, and interaction with, a UIW_SCROLL_BAR class object:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is 1/10th the size of a normal cell by default.

  WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object. A vertical scroll bar will be placed along the right edge of its parent while a horizontal scroll bar will be placed along the bottom edge of its parent.

WOF_SUPPORT_OBJECT—Causes the object to be placed in the parent object's support list. The support list is reserved for objects that are not displayed as part of the user region of the window. If the WOF_-SUPPORT_OBJECT flag is set, the slider will send scroll events to the object it is attached to. If this flag is not set, it will attempt to call its user function. For a slider, this flag must not be set.

- *woAdvancedFlags* are flags (common to all window objects) that determine the advanced operation of the slider object. The following flags (declared in **UI_-WIN.HPP)** control the advanced operation of a slider object:

  WOAF_NO_FLAGS—Does not associate any special advanced flags with the window object. This flag should not be used in conjunction with any other WOAF flags.

  WOAF_NON_CURRENT—Prevents the object from becoming current. If the object is an action object (i.e., it performs an action when selected with the mouse—examples would include a button or a slider) it can still be operated, but will not become current.

- *userFunction*$_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. The user function will be called by the library when:

  1—the user moves onto the object,

  2—the mouse is clicked on the object,

  3—a key was pressed that updates the slider value,

  4—the user moves to a different field in the window or to a different window.

  Because the user function is called at these times, the programmer can do data validation or any other type of necessary operation. The definition of the *userFunction* is as follows:

```
EVENT_TYPE FunctionName(UI_WINDOW_OBJECT *object,
    ULEVENT &event, EVENT_TYPE ccode);
```

returnValue$_{out}$ indicates if an error has occurred. *returnValue* should be 0 if the no error occurred. Otherwise, the programmer should call the error system with an appropriate error message and return -1.

object$_{in}$ is a pointer to the object for which the user function is being called. This argument must be typecast by the programmer if class-specific members need to be accessed.

event$_{in}$ is the run-time message passed to the object.

ccode$_{in}$ is the logical or system code that caused the user function to be called. This code (declared in **UI_WIN.HPP)** will be one of the following constant values:

L_SELECT—The slider was clicked on with the mouse or the end-user pressed a key that updated the slider's value. If this *ccode* is received, *EVENT_TYPE* will have one of the following values:

L_BOTTOM—Indicates that the slider value is being set to its minimum value.

L_CONTINUE_SELECT—Indicates that the slider thumb button is being dragged.

L_DOWN—Indicates that the slider value is decreasing by one interval. This will occur if the down arrow or left arrow key is pressed.

L_END_SELECT—Indicates that the slider thumb button is finished being dragged.

L_PGDN—Indicates that the slider value is decreasing by one interval. This will occur if the <Page Down> key is pressed or if the slider shaft is clicked below or to the left of the thumb button.

L_PGUP—Indicates that the slider value is increasing by one interval. This will occur if the <Page Up> key is pressed or if the slider shaft is clicked above or to the right of the thumb button.

L_TOP—Indicates that the slider value is being set to its maximum value.

L_UP—Indicates that the slider value is increasing by one interval. This will occur if the up arrow or right arrow key is pressed.

S_CURRENT—The object just received focus because the user moved to it from another field or window.

S_NON_CURRENT—The object just lost focus because the user moved to another field or window.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Add a text field to the window.
    UIW_WINDOW *window =
        UIW_WINDOW::Generic(0, 0, 40, 10, "Hello World Window");
    *window
        + &(*new UIW_TEXT(0, 0, 0, 0, "Hello, World!", 1024,
            WNF_NO_FLAGS, WOF_NON_FIELD_REGION)
            + new UIW_SCROLL_BAR(0, 0, 0, 0, SBF_CORNER)
            + new UIW_SCROLL_BAR(0, 0, 0, 0, SBF_VERTICAL)
            + new UIW_SCROLL_BAR(0, 0, 0, 0, SBF_HORIZONTAL));
    *windowManager + window;



    // The scroll bar will automatically be destroyed when the text field
    // is destroyed.
}
```

UIW_SCROLL_BAR::~UIW_SCROLL_BAR

Syntax

```
#include <ui_win.hpp>

virtual ~UIW_SCROLL_BAR(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_SCROLL_-BAR object.

UIW_SCROLL_BAR::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to *_className.*

UIW_SCROLL_BAR::DrawItem

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This virtual advanced function is used to draw the object. If the WOS_OWNERDRAW status is set for the object, this function will be called when drawing the scroll bar or slider. This allows the programmer to derive a new class from UIW_SCROLL_BAR and handle the drawing of the scroll bar or slider, if desired.

- $returnValue_{out}$ is a response based on the success of the function call. If the object is drawn, the function returns a non-zero value. If the object is not drawn, 0 is returned.

- $event_{in}$ contains the run-time message that caused the object to be redrawn. *event.region* contains the region in need of updating. The following logical events may be sent to the DrawItem( ) function:

    S_CURRENT, S_NON_CURRENT, S_DISPLAY_ACTIVE and S_DISPLAY_INACTIVE—Cause the object to be redrawn.

    WM_DRAWITEM—Causes the object to be redrawn. This message is specific to Windows and OS/2.

    Expose—Causes the object to be redrawn. This message is specific to Motif.

- $ccode_{in}$ contains the logical interpretation of *event.*

UIW_SCROLL_BAR::Event

Syntax

include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the scroll bar or slider object. It is declared virtual so that any derived scroll bar class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the scroll bar or slider object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event():**

  L_BEGIN_SELECT—Indicates that the end-user began the selection of the object by pressing the mouse button down while on the object.

  L_DOWN—Causes the slider to decrement its value by one interval. This message is interpreted from a keyboard event.

  L_LEFT—Causes the slider to decrement its value by one interval. This message is interpreted from a keyboard event.

  L_RIGHT—Causes the slider to increment its value by one interval. This message is interpreted from a keyboard event.

L_UP—Causes the slider to increment its value by one interval. This message is interpreted from a keyboard event.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_CURRENT—Causes the object to draw itself to appear current. This message is sent by the Window Manager to a window when it becomes current. The window, in turn, passes this message to the object on the window that is current.

S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager sends this message to a window when the window is subtracted from the Window Manager. The window, in turn, relays the message to all objects attached to it.

S_DISPLAY_ACTIVE—Causes the object to draw itself to appear active. An active object is one that is on the active (i.e., current) window. Most objects do not display differently whether they are active or inactive. An active object should not be confused with a current object. An object is active if it is on the active window. However, it may not be the current object on the window.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed by the event overlaps the region of the object. This message is sent by a UIW_WINDOW to all the non-current, active objects attached to it.

S_DISPLAY_INACTIVE—Causes the object to draw itself to appear inactive. An inactive object is one that is not on the active (i.e., current) window. Most objects do not display differently whether they are inactive or active.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to

redisplay when the region passed with the event overlaps the region of the object. This message is sent by a UIW_WINDOW to all the objects attached to it.

S_HSCROLL—Causes the thumb button of the horizontal scroll bar or slider to be re-positioned, *event.scroll.delta* contains the change in value for the scroll bar or slider object.

S_HSCROLL_SET—Sets the scroll information for the horizontal scroll bar or slider. The thumb button location will be updated to reflect the values. *event, scroll* will contain the new scroll information.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_MOVE—Causes the object to update its location. The distance to move is contained in the *position* field of UI_EVENT. For example, an *event.position.-line* of -10 and an *event.position.column* of 15 moves the object 10 lines up and 15 columns to the right.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another field or window.

S_REDISPLAY—Causes the object to redraw.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

S_VSCROLL—Causes the thumb button of the vertical scroll bar or slider to be re-positioned, *event.scroll.delta* contains the change in value for the scroll bar or slider object.

S_VSCROLL_SET—Sets the scroll information for the vertical scroll bar or slider. The thumb button location will be updated to reflect the values. *event.scroll* will contain the new scroll information.

All other events are passed by **Event( )** to UIW_WINDOW**::Event()** for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own

messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event()** function.

UIW_SCROLL_BAR::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request*, void *"data,*
    ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the scroll bar or slider:

    I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

    I_CLEAR_FLAGS—Clears the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that

contains the flags to be cleared, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to clear the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_-OBJECT. If the SBF_FLAGS are to be cleared, *objectID* should be ID_-SCROLL_BAR. This allows the object to process the request at the proper level. This request only clears those flags that are passed in; it does not simply clear the entire field.

I_GET_FLAGS—Requests the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to obtain the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the SBF_FLAGS are desired, *objectID* should be ID_SCROLL_BAR. This allows the object to process the request at the proper level.

I_GET_VALUE—Returns the scroll bar's or slider's current value. If this message is sent, *data* must be a pointer to a variable of type **int** where the object's value will be copied.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_FLAGS—Sets the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be set, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to set the WOF_-FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the SBF_-FLAGS are to be set, *objectID* should be ID_SCROLL_BAR. This allows the object to process the request at the proper level. This request only sets those flags that are passed in; it does not clear any flags that are already set.

I_SET_VALUE—Sets the scroll bar's or slider's current value. If this message is sent, *data* must be a pointer to a variable of type **int** that contains the object's new value.

All other requests are passed by **Information()** to **UIW_WINDOW::Information()** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include <ui_win.hpp>

ExampleFunction()
{
    SBF_FLAGS flags;
    scrollBar-information (I_GET_FLAGS, &flags, ID_SCROLL_BAR) ;


}
```

Storage Members

This section describes those class members that are used for storage purposes.

UIW_SCROLL_BAR::UIW_SCROLL_BAR

Syntax

#include <ui_win.hpp>

UIW_SCROLL_BAR(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));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_SCROLL_BAR by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a scroll bar or slider is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_SCROLL_BAR::Load

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_SCROLL_BAR from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_m$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_SCROLL_BAR::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_SCROLL_BAR::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's New() function.

- *returnValue*$_{out}$ is a pointer to the object's New() function.

UIW_SCROLL_BAR::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

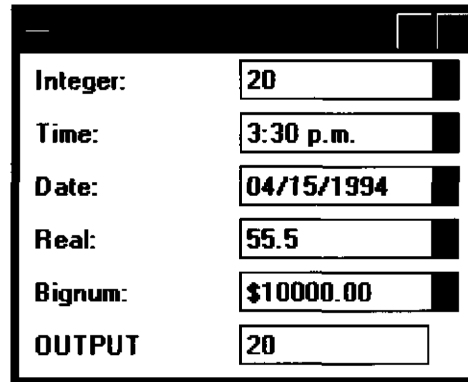
- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see

the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:.-userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 21 - UIW SPIN CONTROL

The UIW_SPIN_CONTROL class object is used to allow the end-user to select a value from a finite range of values. The spin control can be used to set integer, real and bignum values, as well as times and dates. In addition, any user-derived object can be used with the spin control as long as the object processes the I_DECREMENT_VALUE and I_INCREMENT_VALUE requests in its Information() function. The object is called a spin control because it can be thought of as a wheel with the set of values arranged around the outside of the wheel. The end-user spins the wheel bringing each possible value into view. The end-user can type a value into the field or he can spin through the values until the desired value is displayed. The figure below shows a graphical implementation of a UIW_SPIN_CONTROL class object:



The UIW_SPIN_CONTROL class is declared in UI_WIN.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_SPIN_CONTROL : public UIW_WINDOW
{
public:
    static ZIL_ICHAR _className[];
    static int defaultInitialized;

    UIW_SPIN_CONTROL(int left, int top, int width,
        UI_WINDOW_OBJECT *fieldObject, ZIL_INT32 _delta = 1,
        WNF_FLAGS wnFlags = WNF_NO_FLAGS,
        WOF_FLAGS woFlags = WOF_NO_FLAGS,
        ZIL_USER_FUNCTION .userFunction = ZIL_NULLF(ZIL_USER_FUNCTION));
    virtual ~UIW_SPIN_CONTROL(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);
#if defined(ZIL_MOTIF)
    virtual void RegionMax(UI_WINDOW_OBJECT *object);
#endif
```

```
#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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),
        const ZIL_ICHAR *languageName = ZIL_NULLP(ZIL_ICHAR));
    UIW_SPIN_CONTROL(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),
        const ZIL_ICHAR *1anguageName = ZIL_NULLP(ZIL_ICHAR));
    virtual void Load(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));
#endif
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name,
        ZIL_STORAGE *file = ZIL_NULLP(ZIL_STORAGE),
        ZIL_STORAGE_OBJECT *object = ZIL_NULLP(ZIL_STORAGE_OBJECT),
        UI_ITEM *objectTable = ZIL_NULLP(UI_ITEM),
        UI_ITEM *userTable = ZIL_NULLP(UI_ITEM));
#endif
```

General Members

This section describes those members that are used for general purposes.

_*className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_SPIN_CONTROL class, _*className* is "UIW_SPIN_CONTROL."

- *defaultlnitialized* indicates if the default decorations (i.e., images) for this object have been set up. The default decorations are located in the file **IMG_DEF.CPP.** If *defaultlnitialized* is TRUE, the decorations have been set up. Otherwise they have not been.

UIW_SPIN_CONTROL::UIW_SPIN_CONTROL

Syntax

#include <ui_win.hpp>

UIW_SPIN_CONTROL(int *left,* int *top,* int *width,*
UI_WINDOW_OBJECT **fieldObject,* ZIL_INT32 _*delta* = 1,

WNF_FLAGS *wnFlags* = WNF_NO_FLAGS,
WOF_FLAGS *woFlags* = WOF_NO_FLAGS,
ZIL_USER_FUNCTION *_userFunction* = ZIL_NULLF(ZIL_USER_FUNCTION));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_SPIN_CONTROL class object.

- $left_{in}$ and top_{in} is the starting position of the spin control field within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the spin control. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value. The height of the spin control is determined automatically by the UIW_SPIN_CONTROL object.

- $fieldObject_{in}$ is the object whose value is to be set. This can be a pointer to a UIW_BIGNUM, UIWJDATE, UIW_INTEGER, UIW_REAL, UIW_TIME or any user-derived object that processes I_DECREMENT_VALUE and I_INCREMENT_-VALUE requests in its **Information( )** function. This object will be destroyed when the spin control is destroyed. *fieldObject* must have its *range* member set. The *range* defines the acceptable values that can be spun into view. The *range* can consist of multiple ranges, which will be spun through in the proper numerical or chronological order. If the *range* is an open-ended range, the WNF_NO_WRAP flag should be set on the spin control.

- $_delta_{in}$ is the value by which the *fieldObject* value will be adjusted when the object is spun. *_delta* is simply added to or subtracted from *fieldObject* value, so *_delta* should be specified in units appropriate to *fieldObjecf* s type. For example, if *fieldObject* is a UIW_TIME object, *_delta* must be given in hundredths of seconds. When the spin control is spun up or down, the I_DECREMENT_VALUE or I_IN-CREMENTJVALUE request is sent to the *fieldObjecf* s **Information( )** function. *_delta* is passed to the **Information( )** function as the *data* parameter.

- *wnFlags*$_{in}$ are flags that define the operation of the spin control. The following flags (declared in **UI_WIN.HPP)** control the general presentation of the spin control:

  WNF_NO_FLAGS—Does not associate any special flags with the spin control. This flag should not be used in conjunction with any other WNF flags.

  WNF_NO_WRAP—Will not allow spinning up or down to wrap from the maximum value in the range to the lowest or vice versa. This flag should be set if the object has an open-ended range.

- *woFlags*$_{in}$ are flags (common to all window objects) that determine the general operation of the spin control object. The following flags (declared in **UI_WIN.HPP)** control the general presentation of, and interaction with, a UIW_SPIN_CONTROL class object:

  WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags.

  WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor select any menu items. Typically, the object will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

  WOF_VIEW_ONLY—Prevents the object from being edited. However, the object may become current and the user may scroll through the data, mark it, and copy it.

- *_userFunction*$_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. The user function will be called by the library when:

  1—the user moves onto the field,

  2—the <ENTER> key is pressed while the field is current, or

  3—the user moves to a different field in the window or to a different window.

  Because the user function is called at these times, the programmer can do data validation or any other type of necessary operation. The definition of the *userFunction* is as follows:

EVENT_TYPE FunctionName(UI_WINDOW_OBJECT **object,*
UI_EVENT *&event,* EVENT_TYPE *ccode);*

returnValue$_{out}$ indicates if an error has occurred. *returnValue* should be 0 if the no error occurred. Otherwise, the programmer should call the error system with an appropriate error message and return -1.

object$_{in}$ is a pointer to the object for which the user function is being called. This argument must be typecast by the programmer if class-specific members need to be accessed.

event$_{in}$ is the run-time message passed to the object.

ccode$_{in}$ is the logical or system code that caused the user function to be called. This code (declared in **UI_WIN.HPP)** will be one of the following constant values:

L_SELECT—The <ENTER> key was pressed while the field was current.

S_CURRENT—The object just received focus because the user moved to it from another field or window. This code is sent before any editing operations are permitted.

S_NON_CURRENT—The object just lost focus because the user moved to another field or window.

UIW_SPIN_CONTROL::~UIW_SPIN_CONTROL

Syntax

```
#include <ui_win.hpp>

virtual ~UIW_SPIN_CONTROL(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_SPIN_-CONTROL object.

UIW_SPIN_CONTROL::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to *_className.*

UIW_SPIN_CONTROL::Event

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the spin control object. It is declared virtual so that any derived spin control class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event*. If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the spin control object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event( ):**

  L_BEGIN_SELECT—Indicates that the end-user began the selection of the object by pressing the mouse button down while on the object.

  L_CONTINUE_SELECT—Indicates that the end-user previously clicked down on the object with the mouse and is now continuing to hold the mouse button down while on the object.

  L_DOWN—Causes the displayed value to be decremented.

  L_END_SELECT—Indicates that the selection process, initiated with the L_-BEGIN_SELECT message, is complete. For example, the end-user has pressed and released the mouse button.

  L_SELECT—Indicates that the object has been selected. The selection may be the result of a mouse click or a keyboard action.

  L_UP—Causes the displayed value to be incremented.

  L_VIEW—Indicates that the mouse is being moved over the object. This message allows the object to alter the mouse image.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_CURRENT—Causes the object to draw itself to appear current. This message is sent by the Window Manager to a window when it becomes current. The window, in turn, passes this message to the object on the window that is current.

S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager sends this message to a window when the window is subtracted from the Window Manager. The window, in turn, relays the message to all objects attached to it.

S_DISPLAY_ACTIVE—Causes the object to draw itself to appear active. An active object is one that is on the active (i.e., current) window. Most objects do not display differently whether they are active or inactive. An active object should not be confused with a current object. An object is active if it is on the active window. However, it may not be the current object on the window.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed by the event overlaps the region of the object. This message is sent by a UIW_WINDOW to all the non-current, active objects attached to it.

S_DISPLAY_INACTIVE—Causes the object to draw itself to appear inactive. An inactive object is one that is not on the active (i.e., current) window. Most objects do not display differently whether they are inactive or active.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed with the event overlaps the region of the object. This message is sent by a UIW_WINDOW to all the objects attached to it.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_MOVE—Causes the object to update its location. The distance to move is contained in the *position* field of UI_EVENT. For example, an *event.position.-line* of -10 and an *event.position.column* of 15 moves the object 10 lines up and 15 columns to the right.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another field or window.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

S_RESET_DISPLAY—Changes the display to a different resolution, *event.data* should point to the new display class to be used. If *event.data* is NULL, then a text mode display will be created. This event is specific to DOS and must be placed on the event queue by the programmer. The library will never generate this event.

All other events are passed by Event() to UIW_WINDOW::Event() for processing.

UIW_SPIN CONTROL:information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request,* void *"data,*
ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in UI_WIN.HPP) are recognized by the spin control:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_COPY_TEXT—Copies the *text* associated with the object into a buffer provided by the programmer. If this request is sent, *data* must be the address of a buffer where the string's text will be copied. This buffer must be large enough to contain all of the characters associated with the field and the terminating NULL character.

  I_GET_TEXT—Returns a pointer to the text associated with the object. If this request is sent, *data* should be a doubly-indirected pointer to ZIL_ICHAR. This request does not copy the text into a new buffer.

  I_GET_VALUE—Returns the *value* associated with the field. If this message is sent, *data* must be a pointer to a variable where the field's value will be copied. This request is processed by *fieldObject,* so the returned value is specific to *fieldObjecf* s type.

  I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

  I_SET_TEXT—Sets the text associated with the object. This request will also redisplay the object with the new text, *data* should be a pointer to the new text.

  I_SET_VALUE—Sets the *value* associated with the field. If this message is sent, *data* must be a pointer to a variable that contains the field's new value.

This request is processed by *fieldObject,* so the value is specific to *fieldObject's* type.

All other requests are passed by **Information()** to **UIW_WINDOW::Information()** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information( )** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include <ui_win.hpp>

ExampleFunction()
{
    UIF_FLAGS wnflags;
    spinControl->Information(I_GET_FLAGS, &wnflags, ID_WINDOW);



}
```

Storage Members

This section describes those class members that are used for storage purposes.

UIW_SPIN_CONTROL::UIW_SPIN_CONTROL

Syntax

#include <ui_win.hpp>

UIW_SPIN_CONTROL(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));

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced constructor creates a new UIW_SPIN_CONTROL by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a spin control is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library

will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_SPIN_CONTROL::Load

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced function is used to load a UIW_SPIN_CONTROL from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see

the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_SPIN_CONTROL::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1*.

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1*.

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1*. If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1*. If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_SPIN_CONTROL::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual function returns a pointer to the object's New() function.

returnValue$_{out}$ is a pointer to the object's New() function.

UIW_SPIN_CONTROL:-.Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

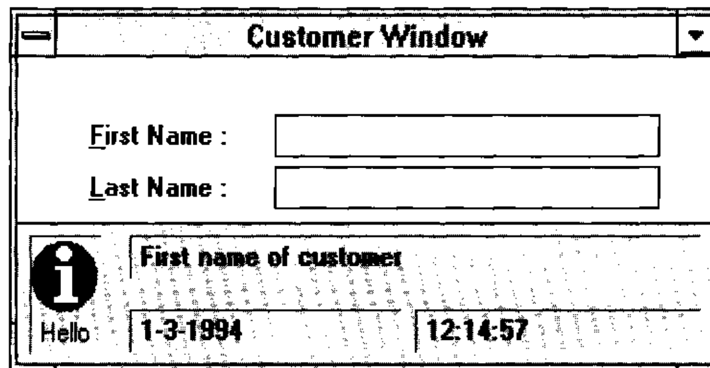
- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UIW_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 22 - UIW_STATUS_BAR

The UIW_STATUS_BAR class object is used to present status information to the end-user. The status bar appears at the bottom of the window to which it is attached. It always covers the entire width of the window. Any object that has a **Drawltem()** function can be displayed on the status bar. This includes string, date, time and number fields; icons; buttons and other derived objects. The status bar is not an interactive object. The objects on the status bar are intended to present information to the end-user, not to obtain feedback from them. Geometry management constraints can be placed on objects on the status bar, if desired. The figure below shows a graphical implementation of a UIW_STATUS_BAR class object with various window objects:



The UIW_STATUS_BAR class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class UIW_STATUS_BAR : public UIW_WINDOW
{
public:
    static ZIL_ICHAR _className[];

    UIW_STATUS_BAR(int _height = 1,
        WOF_FLAGS _woFlags = WOF_BORDER | WOF_SUPPORT_OBJECT);
    virtual ~UIW_STATUS_BAR(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Drawltem(const UI_EVENT &event, EVENT_TYPE ccode);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

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

protected:
    int height;
};
```

General Members

This section describes those members that are used for general purposes.

_*className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_STATUS_BAR class, _*className* is "UIW_STATUS_BAR."

- *height* is the height of the status bar.

UIW_STATU S_BAR::UIW_STATU S_BAR

Syntax

#include <ui_win.hpp>

UIW_STATUS_BAR(int *height* = 1,
    WOF_FLAGS _*woFlags* = WOF_BORDER | WOF_SUPPORT_OBJECT);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_STATUS_BAR class object.

- *height*$_{in}$ is the height of the status bar. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- *_woFlags*$_{in}$ are flags (common to all window objects) that determine the general operation of the status bar object. The following flags (declared in **UI_WIN.HPP)** control the general presentation of a UIW_STATUS_BAR class object:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is l/10th the size of a normal cell by default.

  WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags.

  WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object. The status bar is always a non-field region object, even if this flag is not set explicitly.

  WOF_SUPPORT_OBJECT—Causes the object to be placed in the parent object's support list. The support list is reserved for objects that are not displayed as part of the user region of the window. The user region is the area of the window framed by, but not including, the support objects. If this flag is set the status bar will not be scrolled or overwritten. If the flag is not set, and the window is scrolled, the status bar will scroll with the window and objects on the window may overwrite the status bar if they overlap. This flag is set by default in the constructor.

UIW_STATUS_BAR::~UIW_STATUS_BAR

Syntax

```
#include <ui_win.hpp>

virtual ~UIW_STATUS_BAR(void);
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_STATUS. BAR object. All objects attached to the status bar will also be destroyed.

UIW_STATUS_BAR::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This virtual function returns the class name associated with the object.

- *returnValue*$_{out}$ is a pointer to the *_className* member.

UIW_STATUS_BAR::DrawItem

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual advanced function is used to draw the object. If the WOS_OWNERDRAW status is set for the object, this function will be called when drawing the status bar. This allows the programmer to derive a new class from UIW_STATUS_BAR and handle the drawing of the status bar, if desired. The status bar has the WOS_OWNERDRAW status set by default.

- *returnValue*$_{out}$ is a response based on the success of the function call. If the object is drawn the function returns a non-zero value. If the object is not drawn, 0 is returned.

- *event*$_{in}$ contains the run-time message that caused the object to be redrawn. *event.region* contains the region in need of updating. The following logical events may be sent to the **DrawItem(**) function:

  S_CURRENT, S_NON_CURRENT, S_DISPLAY_ACTIVE and **S_DISPLAY_INACTIVE**—Messages that cause the object to be redrawn.

  WM_DRAWITEM—A message that causes the object to be redrawn. This message is specific to Windows and OS/2.

Expose—A message that causes the object to be redrawn. This message is specific to Motif.

- $ccode_{in}$ contains the logical interpretation of *event.*

UIW_STATUS_BAR::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the status bar object. It is declared virtual so that any derived status bar class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the status bar object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event( ):**

  S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

  S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system.

This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_MOVE—Causes the object to update its location. The distance to move is contained in the *position* field of UI_EVENT. For example, an *event.position.-line* of -10 and an *event.position.column* of 15 moves the object 10 lines up and 15 columns to the right.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

All other events are passed by **Event( )** to **UIW_WINDOW::Event( )** for processing.

UIW STATUS_BAR::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request,* void **data,*
ZIL_OBJECTID *objectID* = IDJDEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the status bar:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

  All other requests are passed by **Information( )** to **UIW_WINDOW::Information( )** for processing.

- *data*$_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- *objectID*$_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include <ui_win.hpp>

ExampleFunction()
{
    UIF_FLAGS woflags;
    statusBar-information(I_GET_FLAGS, &woflags, ID_WINDOW_OBJECT);



}
```

Storage Members

This section describes those class members that are used for storage purposes.

UIW_STATUS_BAR::UIW_STATUS_BAR

Syntax

#include <ui_win.hpp>

UIW_STATUS_BAR(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));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_STATUS_BAR by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a status bar is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter

69—ZIL_STORAGE_OBJECT_READ_ONLY" *of Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_STATUS_BAR::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_STATUS_BAR from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_STATUS_BAR::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UIJWIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_STATUS_BAR::NewFunction

Syntax

#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's New( ) function.

- $returnValue_{out}$ is a pointer to the object's New( ) function.

UIW STATUS BAR::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- $name_{in}$ is the name of the object to be stored.

- $file_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 23 - UIW STRING

The UIW_STRING class is used to display string information and to collect information, in string form, from an end-user. The figure below shows a graphical implementation of a UIW_STRING object:

The UIW_STRING class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_STRING : public UI_WINDOW_OBJECT
{
public:
    static ZIL_ICHAR _className[];
    STF_FLAGS stFlags;
    int insertMode;

    UIW_STRING(int left, int top, int width, ZIL_ICHAR *text,
        int maxLength = -1,
        STF_FLAGS stFlags = STF_NO_FLAGS,
        WOF_FLAGS woFlags = WOF_BORDER | WOF_AUTO_CLEAR,
        ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION));
    virtual ~UIW_STRING(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    ZIL_ICHAR *DataGet(void);
    void DataSet(ZIL_ICHAR *text, int maxLength = -1);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

#if defined (ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
```

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

protected:
    int maxLength;
    ZIL_ICHAR *text;

    virtual EVENT_TYPE Drawltem(const UI_EVENT kevent, EVENT_TYPE ccode);
    ZIL_ICHAR *ParseRange(ZIL_ICHAR *buffer, ZIL_ICHAR *minValue,
        ZIL_ICHAR *maxValue);
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_STRING class, *_className* is "UIW_STRING."

- *stFlags* are flags that define the operation of the UIW_STRING class. A full description of the string flags is given in the UIW_STRING constructor.

- *insertMode* indicates whether the string is in insert or overstrike mode. If *insertMode* is TRUE, the string is in insert mode. Otherwise the string is in overstrike mode.

- *maxLength* is the maximum length of the string buffer. *maxLength* does not include the NULL terminator.

- *text* is the text that is displayed in the string.

UIW_STRING::UIW_STRING

Syntax

#include <ui_win.hpp>

UIW_STRING(int *left,* int *top,* int *width,* ZIL_ICHAR **text,* int *maxLength* = *-1,*
STF_FLAGS *stFlags* = STF_NO_FLAGS,
WOF_FLAGS *woFlags* = WOF_BORDER | WOF_AUTO_CLEAR,
ZIL_USER_FUNCTION *userFunction* = ZIL_NULLF(ZIL_USER_FUNCTION));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_STRING class object.

- $left_{in}$ and top_{in} is the starting position of the string field within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the string field. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value. The height of the string field is determined automatically by the UIW_STRING object.

  $text_{in}$ is the text that is shown on the string. Unless the WOF_NO_ALLOCATE_-DATA flag is set, the text is copied into a buffer, allocated by the UIW_STRING object, which is *maxLength* + 1 characters in length. If the WOF_NO_-ALLOCATE J)ATA flag is set, *text* must be space, allocated by the programmer, that is not deleted until the string field is deleted.

- $maxLength_{in}$ is the maximum length of the string buffer, excluding the NULL terminator. The UIW_STRING object will automatically allocate extra space for the NULL terminator. If *maxLength* is -1 (default value), the size of the string buffer allocated is the initial length of *text,* including the NULL terminator.

- *stFlags*$_{in}$ describes how the string should display. The following flags (declared in UI_WIN.HPP) control the general presentation of a UIW_STRING class object:

  STF_LOWER_CASE—Displays all characters in lowercase.

  STF_NO_FLAGS—Does not associate any special flags with the string object. This flag should not be used in conjunction with any other STF flags.

  STF_PASSWORD—Prevents characters from being echoed to the field. Instead, displayed characters will be an operating system-specific password character (e.g., '*').

  STF_UPPER_CASE—Displays all characters in uppercase.

  STF_VARIABLE_NAME—Converts spaces to underscores (i.e., '_').

- *woFlags*$_{in}$ are flags (common to all window objects) that determine the general operation of the string object. The following flags (declared in UI_WIN.HPP) affect the operation of the UIW_STRING class object:

  WOF_AUTO_CLEAR—Automatically marks the entire buffer if the end-user tabs to the field from another object. If the user then enters data (without first having pressed any movement or editing keys) the entire field will be replaced. This flag is set by default in the constructor.

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOF_JUSTIFY_CENTER—Center-justifies the data within the displayed field.

  WOF_JUSTIFY_RIGHT—Right-justifies the data within the displayed field.

  WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is l/10th the size of a normal cell by default.

WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

WOF_NO_FLAGS—Does not associate any special window flags with the object. Setting this flag left-justifies the data. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor edit the string information. Typically, the field will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

WOF_UNANSWERED—Sets the initial status of the field to be "unanswered." An unanswered field is displayed as an empty field.

WOF_VIEW_ONLY—Prevents the object from being edited. However, the object may become current and the user may scroll through the data, mark it, and copy it.

- *userFunction*$_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. The user function will be called by the library when:

  1—the user moves onto the field,

  2—the <ENTER> key is pressed while the field is current or, if the field is in a list, the mouse is clicked on it, or

  3—the user moves to a different field in the window or to a different window.

  Because the user function is called at these times, the programmer can do data validation or any other type of necessary operation. The definition of the *userFunction* is as follows:

  EVENT_TYPE FunctionName(UI_WINDOW_OBJECT **object,*
  UI_EVENT *&event,* EVENT_TYPE *ccode);*

returnValue$_{out}$ indicates if an error has occurred. *returnValue* should be 0 if the no error occurred. Otherwise, the programmer should call the error system with an appropriate error message and return -1.

object$_{in}$ is a pointer to the object for which the user function is being called. This argument must be typecast by the programmer if class-specific members need to be accessed.

event$_{in}$ is the run-time message passed to the object.

ccode$_{in}$ is the logical or system code that caused the user function to be called. This code (declared in UI_WIN.HPP) will be one of the following constant values:

> **L_SELECT**—The <ENTER> key was pressed while the field was current or, if the field is in a list, the mouse was clicked on the field.
>
> **S_CURRENT**—The object just received focus because the user moved to it from another field or window. This code is sent before any editing operations are permitted.
>
> **S_NON_CURRENT**—The object just lost focus because the user moved to another field or window.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Add a string field to the window.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_STRING(10, 1, 20, "Sample string", 256);



    // The string object will automatically be destroyed when the window
    // is destroyed.
}
```

UIW_STRING::~UIW_STRING

Syntax

```
#include <ui_win.hpp>

virtual ~UIW_STRING(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_STRING object.

UIW_STRING::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to *_className.*

UIW_STRING::DataGet

Syntax

#include <ui_win.hpp>

ZIL_ICHAR *DataGet(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function returns the text information associated with the string object.

- *returnValue*$_{out}$ is a pointer to the text information associated with the string.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_STRING *string)
{
    ZIL_ICHAR *text = string->DataGet();


}
```

UIW_STRING::DataSet

Syntax

#include <ui_win.hpp>

void DataSet(ZIL_ICHAR **text*, int *maxLength* = -1);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function assigns new text to the UIW_STRING object and redisplays the field. If no text is passed in (i.e., *text* is NULL), the field will be redrawn.

- $text_{in}$ is a pointer to the new text. If the WOF_NO_ALLOCATE_DATA flag is set, this argument must be a string, allocated by the programmer, that is not destroyed until the UIW_STRING class object is destroyed. Otherwise, the information associated with this argument is copied by the UIW_STRING class object. If this argument is NULL, no string information is changed, but the string field is redisplayed.

- $maxLength_{in}$ is the number of characters to allocate for the string buffer. If *maxLength* is greater than the string's previous length, a new buffer of size *maxLength* + 1 (for the NULL terminator) is allocated. Otherwise, if *maxLength* is -1 or less than the size of the previous string, the previous buffer is used.

Example

```
#include <ui_win.hpp>
ExampleFunction(UIW_STRING *string)
{



    string->DataSet("Hello World!");
}
```

UIW_STRING::DrawItem

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual advanced function is used to draw the object. If the WOS_OWNERDRAW status is set for the object, this function will be called when drawing the string. This allows the programmer to derive a new class from UIW_STRING and handle the drawing of the string, if desired.

- $returnValue_{out}$ is a response based on the success of the function call. If the object is drawn the function returns a non-zero value. If the object is not drawn, 0 is returned.

- $event_{in}$ contains the run-time message that caused the object to be redrawn. *event, region* contains the region in need of updating. The following logical events may be sent to the **DrawItem( )** function:

  S_CURRENT, S_NON_CURRENT, S_DISPLAY_ACTIVE and **S_DISPLAY_INACTIVE**—Messages that cause the object to be redrawn.

  WM_DRAWITEM—A message that causes the object to be redrawn. This message is specific to Windows and OS/2.

  Expose—A message that causes the object to be redrawn. This message is specific to Motif.

- $ccode_{in}$ contains the logical interpretation of *event.*

UIW_STRING::Event

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function processes run-time messages sent to the string object. It is declared virtual so that any derived string class can override its default operation.

- *returnValue*$_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the string object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event( ):**

  E_KEY—Indicates that a key has been pressed. It places the character in the string at the current cursor position unless the string is already *maxLength* characters. This message is interpreted from a keyboard event.

  L_BACKSPACE—Causes the first editable character to the left of the cursor position to be deleted and moves the cursor to that position. This message is interpreted from a keyboard event.

  L_BEGIN_MARK, L_CONTINUE_MARK and **L_END_MARK—**Communicate the progress of a mark operation. L_BEGIN_MARK indicates that the marking process is beginning, L_CONTINUE_MARK indicates the growth or decrease of the marked region, and L_END_MARK indicates the end of the marking operation. Using a mouse, for example, L_BEGIN_MARK indicates

that the mouse button has been pressed, L_CONTINUE_MARK indicates that the mouse is currently being dragged with the button depressed, and L_END_-MARK indicates that the mouse button has been released. These messages are interpreted from mouse events.

L_BOL—Causes the cursor to move to the beginning of the line. For example, where the underscore represents the cursor position, the string Stand and be counted would change to Stand and be counted as a result of the L_BOL message. If the mark feature is on, L_BOL extends the marked region to the beginning of the line. This message is interpreted from a keyboard event.

L_COPY_MARK—Causes the marked region to be copied into the global paste buffer. This message is interpreted from a keyboard event.

L_CUT—Cuts the marked portion of the string. The cut region is stored in the global paste buffer. This message is interpreted from a keyboard event.

L_DELETE—Causes the marked characters, if any, or the character at the current cursor position, to be deleted. For example, where the underscore represents the cursor position, the string Stand and be counted would change to Stand ad be counted as a result of the L_DELETE message. This message is interpreted from a keyboard event.

L_DELETE_EOL—Causes all characters from the current cursor position to the end of the line to be deleted. For example, where the underscore represents the cursor position, the string Stand and be counted would change to Stand a_ as a result of the L_DELETE_EOL message. This message is interpreted from a keyboard event.

L_DELETE_WORD—Causes the word at the cursor position to be deleted, along with any trailing spaces. For example, where the underscore represents the cursor position, the string Stand and be counted would change to Stand be counted as a result of the L_DELETE_WORD message. This message is interpreted from a keyboard event.

L_END_SELECT—Indicates that the selection process, initiated with the L_-BEGIN_SELECT message, is complete. For example, the end-user has pressed and released the mouse button. The user function will be called.

L_EOL—Causes the cursor to move to the end of the string field. For example, where the underscore represents the cursor position, the string Stand and be counted would change to Stand and be counted_ as a result of the L_EOL

message. If the mark feature is on, L_EOL extends the marked region to the end of the line. This message is interpreted from a keyboard event.

L_INSERT_TOGGLE—Toggles the insert mode. If the current mode is insert mode, any entered character will be inserted into the string at the cursor position. For instance, if the character 'n' were entered, the string Stand ad be counted would change to Stand and be counted.

If the current mode is overstrike mode, any entered character will replace the character at the cursor position. For instance, if the character'd' were entered, the string Stand and be counted would change to Stand add be counted. This message is interpreted from a keyboard event.

L_LEFT—Causes the cursor to move one character or space to the left of its current position, if it is not in the field's first editable position. If the mark feature is on, L_LEFT extends the marked region to include the character. This message is interpreted from a keyboard event.

L_MARK—Turns the mark feature on or off. This message is interpreted from a keyboard event.

L_MARK_BOL—Marks the string from the current cursor position to the beginning of the string and places the cursor at the beginning of the string. This message is interpreted from a keyboard event.

L_MARK_EOL—Marks the string from the current cursor position to the end of the string and places the cursor at the end of the string. This message is interpreted from a keyboard event.

L_MARK_LEFT—Moves the cursor to the left one character, marking the character. This message is interpreted from a keyboard event.

L_MARK_RIGHT—Moves the cursor to the right one character, marking the character. This message is interpreted from a keyboard event.

L_PASTE—Causes the contents of the paste buffer to be placed in the field at the current cursor position. For example, if the contents of the paste buffer were up, the string Stand and be counted would change to Stand up and be counted. If the contents are too large for the string field, only that portion which fits will be pasted. This message is interpreted from a keyboard event.

L_RIGHT—Causes the cursor to move one character or space to the right of its current position, if it is not in the field's last editable position. If the mark

feature is on, L_RIGHT extends the marked region to include the character. This message is interpreted from a keyboard event.

L_SELECT—Indicates that the object has been selected. The selection may be the result of a mouse click or a keyboard action.

L_WORD_LEFT—Causes the cursor position to be moved to the beginning of the current word or, if the cursor is at the beginning of the current word, to the beginning of the next word to the left of the current cursor position. For example, where the underscore represents the cursor position, the string Stand and be counted would change to Stand and be counted, as a result of the L_WORD_LEFT message. This message is interpreted from a keyboard event.

L_WORD_RIGHT—Causes the cursor to move to the beginning of the next word to the right of the current cursor position. For example, where the underscore represents the cursor position, the string Stand and be counted would change to Stand and be counted as a result of the L_WORD_RIGHT message. This message is interpreted from a keyboard event.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_CURRENT—Causes the object to draw itself to appear current. This message is sent by the Window Manager to a window when it becomes current. The window, in turn, passes this message to the object on the window that is current.

S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager sends this message to a window when the window is subtracted from the Window Manager. The window, in turn, relays the message to all objects attached to it.

S_DISPLAY_ACTIVE—Causes the object to draw itself to appear active. An active object is one that is on the active (i.e., current) window. Most objects do not display differently whether they are active or inactive. An active object

should not be confused with a current object. An object is active if it is on the active window. However, it may not be the current object on the window.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed by the event overlaps the region of the object. This message is sent by a UIW_WINDOW to all the non-current, active objects attached to it.

S_DISPLAY_INACTIVE—Causes the object to draw itself to appear inactive. An inactive object is one that is not on the active (i.e., current) window. Most objects do not display differently whether they are inactive or active.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed with the event overlaps the region of the object. This message is sent by a UIW_WINDOW to all the objects attached to it.

S_HSCROLL—Causes the string to scroll its contents. The amount to scroll is contained in *event.scroll.delta.*

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_MOVE—Causes the object to update its location. The distance to move is contained in the *position* field of UI_EVENT. For example, an *event.position.-line* of -10 and an *event.position.column* of 15 moves the object 10 lines up and 15 columns to the right.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another field or window.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

S_VERIFY_STATUS—Causes the object to correlate its state (e.g., cursor position) with the operating system.

All other events are passed by **Event()** to **UI_WINDOW_OBJECT::Event( )** for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event()** function.

UIW_STRING::Information

Syntax

```
#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST request, void *data,
    ZIL_OBJECTID objectID = ID_DEFAULT);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- $returnValue_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- $request_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the string:

    I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

I_CLEAR_FLAGS—Clears the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be cleared, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to clear the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_-OBJECT. If the STF_FLAGS are to be cleared, *objectID* should be ID_-STRING. This allows the object to process the request at the proper level. This request only clears those flags that are passed in; it does not simply clear the entire field.

I_COPY_TEXT—Copies the *text* associated with the object into a buffer provided by the programmer. If this request is sent, *data* must be the address of a buffer where the string's text will be copied. This buffer must be large enough to contain all of the characters associated with the string and the terminating NULL character.

I_GET_FLAGS—Requests the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to obtain the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the STF_FLAGS are desired, *objectID* should be ID_STRING. This allows the object to process the request at the proper level.

I_GET_MAXLENGTH—Gets the *maxLength* value. If *data* is NULL, the address of *maxLength* will be returned. Otherwise, *data* should be a pointer to an integer where *maxLength* will be copied.

I_GET_TEXT—Returns a pointer to the text associated with the object. If this request is sent, *data* should be a doubly-indirected pointer to **ZIL_ICHAR.** This request does not copy the text into a new buffer.

| JNITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_FLAGS—Sets the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be set, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to set the WOF_-FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the STF_-FLAGS are to be set, *objectID* should be ID_STRING. This allows the object

to process the request at the proper level. This request only sets those flags that are passed in; it does not clear any flags that are already set.

I_SET_MAXLENGTH—Sets the *maxLength* member. If this request is sent *data* should be a pointer to an integer that contains the new maximum length.

I_SET_TEXT—Sets the text associated with the object. This request will also redisplay the object with the new text, *data* should be a pointer to the new text.

All other requests are passed by **Information**) to **UI_WINDOW_OBJECT::-Information(**) for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include <ui_win.hpp>

ExampleFunction()
{
    ZIL_ICHAR *text;
    string->Information(I_GET_TEXT, Stext);
    stringl->Information(I_SET_TEXT, "First name");



}
```

UIW_STRING::ParseRange

Syntax

#include <ui_win.hpp>

ZIL_ICHAR *ParseRange(ZIL_ICHAR **buffer*, ZIL_ICHAR **minValue,*
ZIL_ICHAR **maxValue);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function is used to parse one of the ranges passed in to an object derived from UIW_STRING.

- $returnValue_{out}$ indicates where in the range string parsing has progressed to.

- $buffer_{in}$ is a string containing the range.

- $minValue_{out}$ and $maxValue_{out}$ are the minimum and maximum range values that were parsed from the range string pointed to by *buffer.*

Example

```
#include <ui_win.hpp>

int UIW_DATE::Validate(int processError = TRUE)
{

    // Check for an absolute date error.
    ZIL_DATE currentDate;
    ZIL_ICHAR *stringDate = (ZIL_1CHAR *)UIW_STRING::Information(I_GET_TEXT,
        NULL);
    DTI_RESULT errorCode = currentDate.Import(stringDate, dtFlags);

    // Check for a range error,
    if (range && errorCode == DTI_OK)
        errorCode = DTI_OUT_OF_RANGE;
```

```
    for (ZIL_ICHAR *tRange = range; tRange && errorCode == DTI_OUT_OF_RANGE; )
    {
        ZIL_ICHAR minDate[64], maxDate[64];
        tRange = ParseRange(tRange, minDate, maxDate);
        if (currentDate >= ZIL_DATE(minDate, rangeFlags) &&
            currentDate <= ZIL_DATE(maxDate, rangeFlags))
            errorCode = DTI_OK;
    }


    return (errorCode);
}
```

Storage Members

This section describes those class members that are used for storage purposes.

UIW_STRING::UIW_STRING

Syntax

#include <ui_win.hpp>

UIW_STRING(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));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_STRING by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a string object is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

name-to is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:. objectTable* in "Chapter 43—UIJVVIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_STRING::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_STRING from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_STRING::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

objectTable$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

userTable$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_STRING::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's New( ) function.

- *returnValue*$_{out}$ is a pointer to the object's New( ) function.

UIW_STRING "Store

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- $name_{in}$ is the name of the object to be stored.

- $file_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

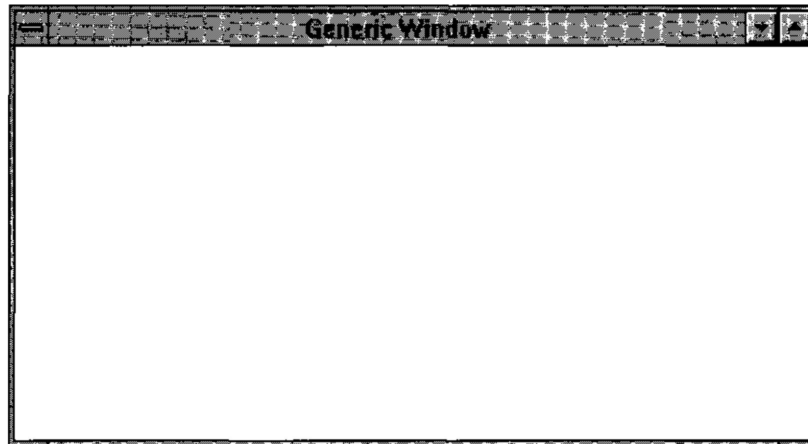
- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the

description of *UI_WINDOW_OBJECT:-.userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 24 - UIW_SYSTEM_BUTTON

The UIW_SYSTEM_BUTTON class is used to provide a small menu of standard, general options that can be performed on a window (e.g., size, move, maximize, minimize, etc.). In addition, clicking on the system button can close the window. The figure below shows a graphical implementation of a UIW_SYSTEM_BUTTON class object (the button with the '—' character):



NOTE: The appearance and operation of the system button varies somewhat across platforms. For example, in OS/2, the system button displays a small representation of the application's minimize icon instead of the dash-like image shown in many other environments.

There are also several significant differences in the operation of the system button on the Macintosh. The first is that the Macintosh does not have the concept of a system button. Instead, it has a close box. If the user clicks on the close box, the window closes. No menu of options will appear. OpenZinc will ignore the menu options and they will have no effect if used in a Macintosh application.

The second operational difference of the UIW_SYSTEM_BUTTON object on the Macintosh is that it can be used to create the Apple menu's "About" item that is common to Macintosh applications. If a UIW_POP_UP_ITEM with the WOF_SUPPORT_OBJECT flag set is added to the UIW_SYSTEM_BUTTON object, that pop-up item will be used as the About item in Macintosh applications. In other environments, the support pop-up item will be ignored by OpenZinc and will have no effect.

The UIW_SYSTEM_BUTTON class is declared in UI_WIN.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_SYSTEM_BUTTON : public UIW_BUTTON
{
public:
    static ZIL_ICHAR _className[];
    static int defaultlnitialized;
    SYF_FLAGS syFlags;
    UIW_POP_UP_MENU menu;

    UIW_SYSTEM_BUTTON(SYF_FLAGS syFlags = SYF_NO_FLAGS);
    UIW_SYSTEM_BUTTON(UI_ITEM *item);
    virtual ~UIW_SYSTEM_BUTTON(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    static UIW_SYSTEM_BUTTON *Generic(void);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_SYSTEM_BUTTON(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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

    // List members.
    UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);
    UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT *object);
    UIW_SYSTEM_BUTTON &operator+(UI_WINDOW_OBJECT *object);
    UIW_SYSTEM_BUTTON koperator-(UI_WINDOW_OBJECT *object);

    void SetDecorations(const ZIL_ICHAR *decorationName);
    void SetLanguage(const ZIL_ICHAR *languageName);

protected:
    const ZIL_LANGUAGE *myLanguage;
    const ZIL_DECORATION *myDecorations;
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_SYSTEM_BUTTON class, *_className* is "UIW_SYSTEM_BUTTON."

- *defaultInitialized* indicates if the default language strings and decorations (i.e., images) for this object have been set up. The default strings are located in the file LANG_DEF.CPP. The default decorations are located in the file IMG_DEF.CPP. If *defaultInitialized* is TRUE, the strings and decorations have been set up. Otherwise they have not been.

- *syFlags* are flags that define the operation of the UIW_SYSTEM_BUTTON class. A full description of the system button flags is given in the UIW_SYSTEM_-BUTTON constructor.

- *menu* is a UIW_POP_UP_MENU that is used to maintain the list of UIW_POP_UP_-ITEMs that serve as the options in the system menu. In most graphical operating systems the display of the system menu is handled by the operating system. In these environments, *menu* is used only to store the data to be presented on the system menu. The data is passed to the operating system to display. If the operating system does not handle displaying a system menu, such as in DOS, then *menu* is actually added to the Window Manager when the system menu is displayed, *menu* has the WOF_BORDER, WOAF_TEMPORARY and WOAF_NO_DESTROY flags set by default.

- *myLanguage* is the ZIL_LANGUAGE object that contains the string translations for this object.

- *myDecorations* is the ZIL_DECORATION object that contains the images for this object.

UIW_SYSTEM_BUTTON::UIW_SYSTEM_BUTTON

Syntax

```
#include <ui_win.hpp>

UIW_SYSTEM_BUTTON(SYF_FLAGS syFlags = SYF_NO_FLAGS);
    or
UIW_SYSTEM_BUTTON(UI_ITEM *item);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These overloaded constructors create a new UIW_SYSTEM_BUTTON class object.

The first overloaded constructor creates a UIW_SYSTEM_BUTTON.

- $syFlags_{in}$ are flags that define the operation of the UIW_SYSTEM_BUTTON class. The following flags (declared in UI_WIN.HPP) control the general presentation of a UIW_SYSTEM_BUTTON class object:

    SYF_NO_FLAGS—Associates no special flags with the system button. If this flag is set, the programmer must add UIW_POP_UP_ITEMs to the system button or no pop-up menu will be displayed when the system button is selected. This is the default argument in the constructor.

    SYF_GENERIC—Creates a generic system button menu. The following pop-up item entries are included in the system button menu:

        Restore—Restores the window from either a maximized or a minimized state.

        Move—Puts the window into a mode that allows the window to be moved.

        Size—Puts the window into a mode that allows the window to be sized.

        Minimize—Minimizes the window.

        Maximize—Maximizes the window.

        Close—Closes the window.

    In addition to the above options, the operating system may place one or more other options in a generic system menu.

The second overloaded constructor creates a UIW_SYSTEM_BUTTON and adds to it UIW_POP_UP_ITEMs created from the UI_ITEM array.

- $item_{in}$ is an array of UI_ITEM structures that will be used to create the UIW_POP_-UP_ITEM structures for the system button's menu. For more information regarding the UI_ITEM structure, see "Chapter 18—UI_ITEM" of *Programmer's Reference Volume 1.*

The system button object is always positioned in the upper left corner of the parent window. To ensure that the system button is drawn correctly, it must be added right after the UIW_MINIMIZE_BUTTON class object. The following example shows the correct order of system button addition.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Create a new window and attach it to the window manager.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_BORDER
        + new UIW_MAXIMIZE_BUTTON
        + new UIW_MINIMIZE_BUTTON
        + &(*new UIW_SYSTEM_BUTTON
            + new UIW_POP_UP_ITEM("&Move", MNIF_MOVE)
            + new UIW_POP_UP_ITEM("&Size", MNIF_SIZE)
            + new UIW_POP_UP_ITEM
            + new UIW_POP_UP_ITEM("&Close", MNIF_CLOSE))
        + new UIW_TITLE("Window 1");
    *windowManager + window;

    // The system button will automatically be destroyed when the window
    // is destroyed.
}
```

UIW_SYSTEM_BUTTON::~UIW_SYSTEM_BUTTON

Syntax

#include <ui_win.hpp>

virtual ~UIW_SYSTEM_BUTTON(void);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_SYSTEM_-BUTTON object.

UIW_SYSTEM_BUTTON::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual function returns the object's class name.

- $returnValue_{out}$ is a pointer to *_className.*

UIW_SYSTEM_BUTTON::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the system button object. It is declared virtual so that any derived system button class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the system button object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event( ):**

  L_BEGIN_SELECT—Indicates that the end-user began the selection of the object by pressing the mouse button down while on the object.

  L_CONTINUE_SELECT—Indicates that the end-user previously clicked down on the object with the mouse and is now continuing to hold the mouse button down while on the object.

  L_SELECT—Indicates that the object has been selected. The selection may be the result of a mouse click or a keyboard action.

  S_ADD_OBJECT—Is used to add an new pop-up item to the receiving system button's menu. A pointer to the new object must be in *event.data.* This message

is interpreted only by those objects that contain a list (e.g., windows, horizontal and vertical lists, combo boxes, etc.).

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager sends this message to a window when the window is subtracted from the Window Manager. The window, in turn, relays the message to all objects attached to it.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_RESET_DISPLAY—Changes the display to a different resolution, *event.data* should point to the new display class to be used. If *event.data* is NULL, a text mode display will be created. This event is specific to DOS and must be placed on the event queue by the programmer. The library will never generate this event.

S_SUBTRACT_OBJECT—Is used to subtract a pop-up item from the receiving system button's menu. A pointer to the pop-up item must be in *event.data.* This message is interpreted only by those objects that contain a list (e.g., windows, horizontal and vertical lists, combo boxes, etc.).

All other events are passed by **Event()** to **UIW_BUTTON::Event( )** for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own

messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event( )** function.

UIW_SYSTEM_BUTTON "Generic

Syntax

#include <ui_win.hpp>

static UIW_SYSTEM_BUTTON *Generic(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function creates a generic system button that has the following options: Restore, Move, Size, Minimize, Maximize and Close. In addition to these options, the operating system may place one or more other options in a generic system menu.

- *returnValue*$_{out}$ is a pointer to the constructed UIW_SYSTEM_BUTTON object.

Example

```
#include <ui_win.hpp>

UIW_WINDOW *UIW_WINDOW::Generic(int left, int top, int width, int height,
    char *title, UI_WINDOW_OBJECT *minObject, WOF_FLAGS woFlags,
    WOAF_FLAGS woAdvancedFlags, int _helpContext)
{
    UIW_WINDOW *window = new UIW_WINDOW(left, top, width, height, _icon,
        woFlags, woAdvancedFlags, _helpContext);

    // Add default window objects.
    *window
        + new UIW_BORDER
        + new UIW_MAXIMIZE_BUTTON
        + new UIW_MINIMIZE_BUTTON
        + UIW_SYSTEM_BUTTON::Generic()
        + new UIW_TITLE(title);
```

```
    // Return a pointer to the new window,
    return  (window);
}
```

UIW_SYSTEM_BUTTON::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request*, void **data,*
ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the system button:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_CLEAR_FLAGS—Clears the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be cleared, and *objectID* should indicate the type of object

with which the flags are associated. For example, if the programmer wishes to clear the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_-OBJECT. If the SYF_FLAGS are to be cleared, *objectID* should be ID_-SYSTEM_BUTTON. This allows the object to process the request at the proper level. This request only clears those flags that are passed in; it does not simply clear the entire field.

I_GET_FLAGS—Requests the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to obtain the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the SYF_FLAGS are desired, *objectID* should be ID_SYSTEM_BUTTON. This allows the object to process the request at the proper level.

I_GET_NUMBERID_OBJECT—Returns a pointer to an object whose *numberlD* matches the value in *data*, if one exists. This object does a depth-first search of the objects attached to it, looking for a match of the *numberlD.* If no object has a *numberlD* that matches *data*, NULL is returned. If this message is sent, *data* must be a pointer to a programmer defined NUMBERID.

I_GET_STRINGID_OBJECT—Returns a pointer to an object whose *stringID* matches the character string in *data,* if one exists. This object does a depth-first search of the objects attached to it looking for a match of the *stringID.* If no object has a *stringID* that matches *data,* NULL is returned. If this message is sent, *data* must be a pointer to a programmer defined string.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_FLAGS—Sets the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be set, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to set the WOF_-FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the SYF_-FLAGS are to be set, *objectID* should be ID_SYSTEM_BUTTON. This allows the object to process the request at the proper level. This request only sets those flags that are passed in; it does not clear any flags that are already set.

All other requests are passed by **Information( )** to **UIW_BUTTON::Information()** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the most derived class.

Example

```
#include <ui_win.hpp>

ExampleFunction()
{
    UIF_FLAGS flags;
    systemButton->Information(I_GET_FLAGS, &flags);


}
```

UIW_SYSTEM_BUTTON::SetDecorations

Syntax

#include <ui_win.hpp>

void SetDecorations(const ZIL_ICHAR **decorationName);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function sets the decorations (i.e., images) to be used by the object. The images for the object will be loaded and the object's *myDecorations* member will be updated to point

to the new ZIL_DECORATION object. By default, the object uses the images identified in the **IMG_DEF.CPP** file, which compiles into the library. (If different default images are desired, simply copy a **IMG_<ISO>.CPP** file from the OpenZinc\SOURCE\INTL directory to the \OpenZinc\SOURCE directory, and rename it to **IMG_DEF.CPP** before compiling the library.) The images are loaded from the **I18N.DAT** file, so it must be shipped with your application.

- *decorationName*$_{in}$ is the two-letter ISO country code identifying which images the object should use.

UIW_SYSTEM_BUTTON::SetLanguage

Syntax

#include <ui_win.hpp>

void SetLanguage(const ZIL_ICHAR **languageName);*

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function sets the language to be used by the object. The string translations for the object will be loaded and the object's *myLanguage* member will be updated to point to the new ZIL_LANGUAGE object. By default, the object uses the language identified in the **LANG_DEF.CPP** file, which compiles into the library. (If a different default language is desired, simply copy a **LANG_<ISO>.CPP** file from the OpenZinc\SOURCE\-INTL directory to the \OpenZinc\SOURCE directory, and rename it to **LANG_DEF.CPP** before compiling the library.) The language translations are loaded from the **I18N.DAT** file, so it must be shipped with your application.

- *languageName*$_{in}$ is the two-letter ISO language code identifying which language the object should use.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_SYSTEM_BUTTON::UIW_SYSTEM_BUTTON

Syntax

```
#include <ui_win.hpp>

UIW_SYSTEM_BUTTON(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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_SYSTEM J5UTTON by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a system button is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter

69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW SYSTEM_BUTTON::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced function is used to load a UIW_SYSTEM_BUTTON from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW SYSTEM BUTTON::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_SYSTEM_BUTTON::NewFunction

Syntax

#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New( )** function.

- *returnValue*$_{in}$ is a pointer to the object's **New()** function.

UIW_SYSTEM_BUTTON::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *$name_{in}$* is the name of the object to be stored.

- *$file_{in}$* is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1*.

- *$object_{in}$* is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1*.

- *$objectTable_{in}$* is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1*. If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *$userTable_{in}$* is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT: .userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1*. If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 25 - UIW_TABLE

The UIW_TABLE class object is used to create a table of records. The table can be scrolled both vertically and horizontally. Records in the table can have one or more fields which may be of different types. A user function can be associated with the table record. Records are created using the UIW_TABLE_RECORD class. See "Chapter 27—UIW_-TABLE_RECORD" for more information.

A spreadsheet type object can be created using the UIW_TABLE class by creating a table with multiple columns. Each cell in the spreadsheet typically consists of one field, but can be made up of multiple fields.

A header describing the rows and columns can be placed at the top, side, or both top and side of the table. Headers are created using the UIW_TABLE_HEADER class. See "Chapter 26—UIW_TABLE_HEADER" for more information.

The figure below shows a graphical representation of a table with a column header, a row header, and a corner header:

The UIW_TABLE class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class UIW_TABLE : public UIW_WINDOW
{
public:
    TBLF_FLAGS tblFlags;

    UIW_TABLE(int left, int top, int width, int height, int columns = 1,
        int recordSize = 0, int maxRecords = -1,
        void *data = ZIL_NULLP(void), int records = 0,
        TBLF_FLAGS tblFlags = TBLF_NO_FLAGS,
        WOF_FLAGS woFlags = WOF_BORDER);
    virtual ~UIW_TABLE(void);
```

```
	int DataSet(void *data, int records = 0, int maxRecords = -1);
	void *DataGet(int *records = ZIL_NULLP(int));

	void InsertRecord(int recordNum, void *data = ZIL_NULLP(void));
	void DeleteRecord(int recordNum);
	void *GetRecord(int recordNum);

	virtual EVENT_TYPE Event(const UI_EVENT &event);
	virtual void *Information(INFO_REQUEST request, void *data,
		OBJECTID objectID = ID_DEFAULT);

#if defined (ZIL_LOAD)
	virtual ZIL_NEW_FUNCTION NewFunction(void) ;
	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));
	UIW_TABLE(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));
	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
#if defined (ZIL_STORE)
	virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
		ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
		UI_ITEM *userTable);
#endif

protected:
	UI_WINDOW_OBJECT *columnHeader;
	UI_WINDOW_OBJECT *rowHeader;
	UI_WINDOW_OBJECT *tableRecord;
	UI_WINDOW_OBJECT *virtualRecord;
	void *data;
	int columns;
	int currentRecord;
	int maxRecords;
	int records;
	int recordSize;
	int topRecord;

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

	void DrawRecord(int recordNum);
	void SetCurrent(int recordNum);
};
```

General Members

This section describes those members that are used for general purposes.

- _className contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_TABLE class, _className is "UIW_TABLE."

- *tblFlags* are flags that define the operation of the UIW_TABLE class. A full description of the table flags is given in the UIW_TABLE constructor.

- *columnHeader* is a pointer to the UIW_TABLE_HEADER object that is placed at the top of the table.

- *rowHeader* is a pointer to the UIW_TABLE_HEADER object that is placed at the side of the table.

- *tableRecord* is a pointer to the UIW_TABLE_RECORD object that manages the fields of each table record. *tableRecord* is manipulated so that it is always used for the current record in the table, thus allowing the end-user to move from field to field in the record and enter data, if desired. *tableRecord* is part of the table's list of objects.

- *virtualRecord* is a copy of *tableRecord*. *virtualRecord* is used to draw those records that are not the current record. To minimize memory requirements, only one UIW_-TABLE_RECORD is added to the table. That table record is maintained by *tableRecord*. *virtualRecord*'s position is repeatedly modified so that it can be used to draw all the records other than *tableRecord*. *virtualRecord* is never added to the table's list of objects.

- *data* is a pointer to the data that is displayed in the table fields.

- *columns* is how many columns are displayed by the table. Columns must be 1, the default, for a table. If columns is greater than 1, then a spreadsheet is created. Each field in a spreadsheet must be the same type object.

- *currentRecord* is the record number of the current record.

- *maxRecords* is how many records can be added to the table. If *maxRecords* is -1, the default, then there is no maximum number of records associated with the table. -1 should be specified only if the table has the WOF_NO_ALLOCATE_DATA flag set.

- *records* is the number of records whose data is present in the *data* parameter.

- *recordSize* is the size of the data used by each record.

- *topRecord* is the record number of the top record currently visible in the table.

UIW_TABLE::UIW_TABLE

Syntax

```
#include <ui_win.hpp>

UIW_TABLE(int left, int top, int width, int height, int columns = 1,
    int recordSize = 0, int maxRecords = -1, void *data = ZIL_NULLP(void),
    int records = 0, TBLF_FLAGS tblFlags = TBLF_NO_FLAGS,
    WOF_FLAGS woFlags = WOF_BORDER);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_TABLE class object.

- $left_{in}$ and top_{in} is the starting position of the table. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the table. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $height_{in}$ is the height of the table. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $columns_{in}$ is how many columns of records should be displayed in the table.

- $recordSize_{in}$ is the size of each record's data.

- *maxRecords*$_{in}$ is the maximum number of records that can be added to the table. If *maxRecords* is -1, the default, then there is no maximum number of records associated with the table. -1 should be specified only if the table has the WOF_NO_-ALLOCATE_DATA flag set.

- *data*$_{in}$ is a pointer to the data for the records to be displayed in the table. This memory will be copied by the UIW_TABLE object unless the WOF_NO_-ALLOCATE_DATA flag is set. If the WOF_NO_ALLOCATE_DATA flag is set, then *data* must not be deleted until the table is deleted.

- *records*$_{in}$ indicates how many records are being used initially.

- *tblFlags*$_{in}$ are flags that determine the general operation of the table. The following flags (declared in **UI_WIN.HPP)** affect the operation of a UIW_TABLE class object:

  TBLF_GRID—Causes vertical and horizontal lines to be displayed between records.

  TBLF_NO_FLAGS—Does not associate any special flags with the UIW_-TABLE class object. This flag should not be used in conjunction with any other TBLF flags. This flag is set by default in the constructor.

- *woFlags*$_{in}$ are flags (common to all window objects) that determine the general operation of the table. The following flags (declared in **UI_WIN.HPP)** affect the operation of a UIW_TABLE class object:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is 1/10th the size of a normal cell by default.

  WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating

the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

UIW_TABLE::UIW_TABLE

Syntax

include <ui_win.hpp>

virtual ~UIW_TABLE(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the UIW_TABLE class object. All objects attached to the table are also destroyed.

UIW_TABLE::DataGet

Syntax

#include <ui_win.hpp>

void *DataGet(int **records* = ZIL_NULLP(int));

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function gets a pointer to the data associated with the table.

- *returnValue*$_{out}$ is a pointer to the data block associated with the table.

- *records*$_{in}$ indicates how many records are present.

UIW_TABLE::DataSet

Syntax

```
#include <ui_win.hpp>

int DataSet(void *data, int records = 0, int maxRecords = -1);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function sets the data associated with the table.

- *returnValue*$_{out}$ indicates if the data was successfully set. *returnValue* is non-zero on success. Otherwise, it is zero.

- *data*$_{in}$ is a pointer to the data for the records to be displayed in the table. This memory will be copied by the UIW_TABLE object unless the WOF_NO_-

ALLOCATE_DATA flag is set. If the WOF_NO_ALLOCATE_DATA flag is set, then *data* must not be deleted until the table is deleted.

- $records_{in}$ indicates how many records are being used.

- $maxRecords_{in}$ is the maximum number of records that can be added to the table.

UIW_TABLE::DeleteRecord

Syntax

```
#include <ui_win.hpp>

void DeleteRecord(int recordNum);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function deletes a record. The table should be redisplayed after deleting a record.

- $recordNum_{in}$ is the number of the record to be deleted. Records are zero-indexed, so the first record is record zero.

UIW_TABLE::DrawItem

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual advanced function is used to draw the object. If the WOS_OWNERDRAW status is set for the object, this function will be called when drawing the table. This allows the programmer to derive a new class from UIW_TABLE and handle the drawing of the table, if desired. For the UIW_TABLE object, the WOS_OWNERDRAW status is always set.

- $returnValue_{out}$ is a response based on the success of the function call. If the object is drawn the function returns a non-zero value. If the object is not drawn, 0 is returned.

- $event_{in}$ contains the run-time message that caused the object to be redrawn. *event.region* contains the region in need of updating. The following logical events may be sent to the **Drawltem( )** function:

  S_CURRENT, S_NON_CURRENT, S_DISPLAY_ACTIYE and **S_DISPLAY_INACTIVE**—Messages that cause the object to be redrawn.

  WM_DRAWITEM—A message that causes the object to be redrawn. This message is specific to Windows and OS/2.

  Expose—A message that causes the object to be redrawn. This message is specific to Motif.

- $ccode_{in}$ contains the logical interpretation of *event.*

UIW TABLE::DrawRecord

Syntax

#include <ui_win.hpp>

void DrawRecord(int *recordNum);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function draws a record. Typically an S_SET_DATA event is sent to the record before it is drawn. The S_SET_DATA event can be handled by a derived UIW_-TABLE_RECORD object or the table object's user function will be called with a ccode of S_SET_DATA. This gives the table record the opportunity to update the data in its fields.

- *recordNum*$_{in}$ is the number of the record to be drawn.

UIW_TABLE::Event

Syntax

```
#include <ui_win.hpp>

virtual EVENT_TYPE Event(const UI_EVENT 8cevent);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the table object. It is declared virtual so that any derived table class can override its default operation.

- *returnValue*$_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the table object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event( ):**

  L_BEGIN_SELECT—Indicates that the end-user began the selection of the object by pressing the mouse button down while on the object. The table will route this message to the object that the mouse event occurred on.

  L_DOWN—Moves the focus down one object. This message is interpreted from a keyboard event.

  L_LEFT—Moves the focus left one object. This message is interpreted from a keyboard event.

  L_RIGHT—Moves the focus right one object. This message is interpreted from a keyboard event.

  L_UP—Moves the focus up one object. This message is interpreted from a keyboard event.

  S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

  S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

  S_HSCROLL—Causes the table to scroll horizontally, *event.scrolldelta* indicates how far to scroll.

  S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

  S_VSCROLL—Causes the table to scroll vertically, *event.scrolldelta* indicates how far to scroll.

All other events are passed by **Event( )** to **UIW_WINDOW::Event()** for processing.

UIW_TABLE::GetRecord

Syntax

#include <ui_win.hpp>

void *GetRecord(int *recordNum);*

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function gets the data for the record specified by *recordNum.*

- *returnValue*$_{out}$ is a pointer to the data associated with the record specified by *recordNum.*

- *recordNum*$_{in}$ is the number of the record whose data is to be returned.

UIW_TABLE::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request*, void **data,*
    ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the table:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_CLEAR_FLAGS—Clears the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be cleared, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to clear the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_-OBJECT. If the TBLF_FLAGS are to be cleared, *objectID* should be ID_-TABLE. This allows the object to process the request at the proper level. This request only clears those flags that are passed in; it does not simply clear the entire field.

  I_GET_COLUMNS—Requests the number of columns in the table. If this request is sent, *data* should be a pointer to a variable of type **int.**

  I_GET_CORNER_HEIGHT—Requests the height to be used for the table's corner header. If this request is sent, *data* should be a pointer to a variable of type **int.**

I_GET_CORNER_WIDTH—Requests the width to be used for the table's corner header. If this request is sent, *data* should be a pointer to a variable of type **int.**

I_GET_FLAGS—Requests the current flag settings, for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to obtain the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the TBLF_FLAGS are desired, *objectID* should be ID_TABLE. This allows the object to process the request at the proper level.

I_GET_HEIGHT—Requests the height of the table. If this request is sent, *data* should be a pointer to a variable of type **int.**

I_GET_RECORDS—Requests the number of records in the table. If this request is sent, *data* should be a pointer to a variable of type **int.**

I_GET_RECORD_HEIGHT—Requests the height of records in the table. If this request is sent, *data* should be a pointer to a variable of type **int.**

I_GET_RECORD_WIDTH—Requests the width of records in the table. If this request is sent, *data* should be a pointer to a variable of type **int.**

I_GET_WIDTH—Requests the width of the table. If this request is sent, *data* should be a pointer to a variable of type **int.**

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_COL_HEADER—Sets the *columnHeader* member to point to the object passed in *data.* This request generally should not be used by the programmer.

I_SET_FLAGS—Sets the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be set, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to set the WOF_-FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the TBLF_FLAGS are to be set, *objectID* should be ID_TABLE. This allows the object to process the request at the proper level. This request only sets those flags that are passed in; it does not clear any flags that are already set.

I_SET_ROW_HEADER—Sets the *rowHeader* member to point to the object passed in *data.* This request generally should not be used by the programmer.

I_SET_TABLE_RECORD—Sets the *tableRecord* member to point to the object passed in *data.* This request generally should not be used by the programmer.

I_SET_VIRTUAL_RECORD—Sets the *virtualRecord* member to point to the object passed in *data.* This request generally should not be used by the programmer.

All other requests are passed by **Information()** to **UIW_WINDOW::Information( )** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information(**) function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

UIW TABLE::InsertRecord

Syntax

#include <ui_win.hpp>

void InsertRecord(int *recordNum,* void **data* = ZIL_NULLP(void));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function inserts a record into the table at the position indicated by *recordNum.* All records after *recordNum* will be shifted down one record number. The table should be redisplayed after inserting a new record.

- $recordNum_{in}$ is the position at which the record should be inserted.

- $data_{in}$ is the data for the new record.

UIW_TABLE::SetCurrent

Syntax

#include <ui_win.hpp>

void SetCurrent(int *recordNum);*

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function sets which record is current.

$recordNum_{in}$ is the record that is to be made current.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_TABLE::UIW_TABLE

Syntax

```
#include <ui_win.hpp>

UIW_TABLE(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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_TABLE by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a table is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- object$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL,

the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TABLE::Load

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIWJABLE from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objects* is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the

programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1*.

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW TABLE::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:-.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TABLE::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New( )** function.

returnValue$_{out}$ is a pointer to the object's **New( )** function.

UIW TABLE::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 26 - UIW TABLE HEADER

The UIW_TABLE_HEADER class object is used to provide information about the data contained in a table column or row. A column header appears at the top of a table, a row header appears down the side of the table, and a corner header appears in the corner between a column and row header. The UIW_TABLE_HEADER class is derived from UIW_TABLE, so its operation is very similar to the UIW_TABLE class.

A UIW_TABLE_RECORD object is added to the table header to define the data to be displayed for each column or row field. When the table header needs to be updated it sends S_SET_DATA events to the table record. The *event. rawCode* is the record number and *event.data* is a pointer to the data for the header's record.

The UIW_TABLE_HEADER class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class UIW_TABLE_HEADER : public UIW_TABLE
{
public:
    THF_FLAGS thFlags;

    UIW_TABLE_HEADER(THF_FLAGS thFlags, int recordSize = 0,
        int maxRecords = -1, void *data = ZIL_NULLP(void),
        WOF_FLAGS woFlags = WOF_BORDER);
    virtual ~UIW_TABLE_HEADER(void);

    virtual EVENT_TYPE Event (const UI_EVENT Seventh-
    virtual void *Information(INFO_REQUEST request, void *data,
        OBJECTID objectID = ID_DEFAULT);

#if defined (ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_TABLE_HEADER(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));
    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

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

```
protected:
    virtual EVENT_TYPE Drawltem(const UI_EVENT &event, EVENT_TYPE ccode);
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_TABLE_HEADER class, *_className* is "UIW_TABLE_HEADER."

- *thFlags* are flags that define the operation of the UIW_TABLE_HEADER class. A full description of the table header flags is given in the UIW_TABLE_HEADER constructor.

UIW_TABLE_HEADER::UIW_TABLE_HEADER

Syntax

#include <ui_win.hpp>

UIW_TABLE_HEADER(THF_FLAGS *thFlags,* int *recordSize* = 0,
int *maxRecords* = -1, void **data* = ZIL_NULLP(void),
WOF_FLAGS *woFlags* = WOF_BORDER);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This constructor creates a new UIW_TABLE_HEADER class object.

- $thFlags_{in}$ are flags that determine the general operation of the table header. The following flags (declared in UI_WIN.HPP) affect the operation of a UIW_TABLE_-HEADER class object:

  THF_COLUMN_HEADER—Causes the header to be created as a column header.

  THF_CORNER_HEADER—Causes the header to be created as a corner header.

  THF_GRID—Causes vertical and horizontal lines to be displayed between records in the header.

  THF_NO_FLAGS—Does not associate any special flags with the UIW_-TABLE_HEADER class object. This flag should not be used in conjunction with any other THF flags.

  THF_ROW_HEADER—Causes the header to be created as a row header.

- $recordSize_{in}$ is the size of each record's data.

- $maxRecords_{in}$ is the maximum number of records that can be added to the table.

- $data_{in}$ is a pointer to the data for the records to be displayed in the table. This memory will be copied by the UIW_TABLE_HEADER object unless the WOF_NO_-ALLOCATE_DATA flag is set. If the WOF_NO_ALLOCATE_DATA flag is set, then *data* must not be deleted until the table header is deleted.

- $woFlags_{in}$ are flags (common to all window objects) that determine the general operation of the table header. The following flags (declared in UI_WIN.HPP) affect the operation of a UIW_TABLE_HEADER class object:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating

the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags.

UIW_TABLE_HEADER::~UIW_TABLE_HEADER

Syntax

```
#include <ui_win.hpp>

virtual ~UIW_TABLE_HEADER(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the UIW_TABLE_HEADER class object. All objects attached to the table header are also destroyed.

UIW_TABLE_HEADER::DrawItem

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual advanced function is used to draw the object. If the WOS_OWNERDRAW status is set for the object, this function will be called when drawing the table header. This allows the programmer to derive a new class from UIW_TABLE_HEADER and handle the drawing of the table header, if desired. For the UIW_TABLE_HEADER object, the WOS_OWNERDRAW status is always set.

- $returnValue_{out}$ is a response based on the success of the function call. If the object is drawn the function returns a non-zero value. If the object is not drawn, 0 is returned.

- $event_{in}$ contains the run-time message that caused the object to be redrawn. *event.region* contains the region in need of updating. The following logical events may be sent to the **DrawItem()** function:

  S_CURRENT, S_NON_CURRENT, S_DISPLAY_ACTIVE and **S_DISPLAY_INACTIVE**—Messages that cause the object to be redrawn.

  WM_DRAWITEM—A message that causes the object to be redrawn. This message is specific to Windows and OS/2.

  Expose—A message that causes the object to be redrawn. This message is specific to Motif.

- $ccode_{in}$ contains the logical interpretation of *event.*

UIW TABLE HEADER::Event

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the table header object. It is declared virtual so that any derived table header class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the table header object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event( ):**

  S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

  S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

  S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

  All other events are passed by **Event(**) to **UIW_TABLE::Event(**) for processing.

UIW_TABLE_HEADER::Information

Syntax

```
#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST request, void *data,
    ZIL_OBJECTID objectID = ID_DEFAULT);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- $returnValue_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- $request_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the table header:

  I_CLEAR_FLAGS—Clears the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be cleared, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to clear the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_-OBJECT. If the THF_FLAGS are to be cleared, *objectID* should be ID_-TABLE_HEADER. This allows the object to process the request at the proper level. This request only clears those flags that are passed in; it does not simply clear the entire field.

  I_GET_FLAGS—Requests the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS, and *objectID* should indicate the type of object with which the flags are associated.

For example, if the programmer wishes to obtain the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the THF_FLAGS are desired, *objectID* should be ID_TABLE_HEADER. This allows the object to process the request at the proper level.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_FLAGS—Sets the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be set, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to set the WOF_-FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the THF_-FLAGS are to be set, *objectID* should be ID_TABLE_HEADER. This allows the object to process the request at the proper level. This request only sets those flags that are passed in; it does not clear any flags that are already set.

All other requests are passed by **Information( )** to **UIW_TABLE::Information( )** for processing.

- $data_{in/ovt}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information( )** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID*, the object will attempt to interpret the request with the *objectID* of the actual object type.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_TABLE_HEADER::UIW_TABLE_HEADER

Syntax

#include <ui_win.hpp>

UIW_TABLE_HEADER(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));

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This advanced constructor creates a new UIW_TABLE_HEADER by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a table header is stored in a data file it is usually stored as part of a UIW_TABLE and will be loaded when the table is loaded.

- *name*$_{in}$ is the name of the object to be loaded.

- file$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- object$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:-.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL,

the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TABLE_HEADER::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced function is used to load a UIW_TABLE_HEADER from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume J.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TABLE_HEADER::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TABLE HEADER::NewFunction

Syntax

#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New( )** function.

$returnValue_{out}$ is a pointer to the object's **New( )** function.

UIW_TABLE_HEADER::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- $name_{in}$ is the name of the object to be stored.

- $file_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 27 - UIW TABLE RECORD

The UIW_TABLE_RECORD class object is used to define the fields for a record in the UIW_TABLE object. The UIW_TABLE_RECORD class is for use with the UIW_-TABLE object only. Objects are added to the table record just like they would be added to a window. The table record is then added to the table. To reduce memory requirements, only one UIW_TABLE_RECORD object should be added to a table. The table record's position and data are manipulated so that the table record is always being used to display the current record. This way, the end-user can interact with the fields of the record. All other visible records are only images on the screen, drawn using a copy of the table record and the data provided to the table.

Data is associated with the fields in the table record at run-time. The table record will be sent an S_SET_DATA event from the UIW_TABLE. The *event.rawCode* field will contain the record number and *event.data* will contain a pointer to the record's data. A derived table record can process the S_SET_DATA event in its **Event()** function. If the UIW_TABLE_RECORD class processes the S_SET_DATA event (i.e., there is not a derived instance of UIW_TABLE_RECORD) then it will call the table record's user function with the above data. The application must place the data in the record fields as required.

The UIW_TABLE_RECORD class is declared in **UI_WIN**.HPP. Its public and protected members are:

```
class UIW_TABLE_RECORD : public UIW_WINDOW
{
public:
    UIW_TABLE_RECORD(int width, int height,
        ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION),
        WOF_FLAGS woFlags = WOF_NO_FLAGS);

    virtual EVENT_TYPE Event (const UI_EVENT Seventh-
    virtual void *Information(INFO_REQUEST request, void *data,
        OBJECTID objectID = ID_DEFAULT);
    virtual void RegionMax(UI._WINDOW_OBJECT *object);

#if defined (ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_TABLE_RECORD(const ZIL_ICHAR *name,
        ZIL_STORAGE_READ_ONLY *file = ZIL_NULLP(ZIL_STORAGE_READ_ONLY),
        ZIL_STORAGE_OBJECT_READ_ONLY *object =
            ZIL_NULL P(ZIL_STORAGE_OBJECT_READ_ONLY),
        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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

protected:
    void *data;
    int recordNum;
    int virtualRecord;

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

    UIW_TABLE_RECORD *VirtualRecord(void);
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_TABLE_RECORD class, *_className* is "UIW_TABLE_RECORD."

- *data* is a pointer to the data associated with the table record.

- *recordNum* is the table record number, record numbers are zero-based, so the first record is record zero.

- *virtualRecord* indicates if the table record is the table's virtual record. If *virtualRecord* is TRUE, the record is the virtual record. Otherwise, it is not.

UIW TABLE_RECORD::UIW TABLE RECORD

Syntax

#include <ui_win.hpp>

UIW_TABLE_RECORD(int *width,* int *height,*
    ZIL_USER_FUNCTION *userFunction* = ZIL_NULLF(ZIL_USER_FUNCTION),
    WOF_FLAGS *woFlags* = WOF_NO_FLAGS);

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This constructor creates a new UIW_TABLE_RECORD class object.

- $width_{in}$ is the width of the record. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $height_{in}$ is the height of the record. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $userFunction_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. The user function will be called by the library when:

  1—the user moves onto the record,

  2—the record is selected, or

  3—the user moves to a different record in the table or to a different object in the window, or

  4—the record needs its fields updated.

  Because the user function is called at these times, the programmer can do data validation or any other type of necessary operation. The definition of the *userFunction* is as follows:

  ```
  EVENT_TYPE FunctionName(UI_WINDOW_OBJECT *object,
      UI_EVENT &event, EVENT_TYPE ccode);
  ```

  $returnValue_{out}$ indicates if an error has occurred. *returnValue* should be 0 if the no error occurred. Otherwise, the programmer should call the error system with an appropriate error message and return -1.

$object_{in}$ is a pointer to the object for which the user function is being called. This argument must be typecast by the programmer if class-specific members need to be accessed.

$event_{in}$ is the run-time message passed to the object.

$ccode_{in}$ is the logical or system code that caused the user function to be called. This code (declared in **UI_EVT.HPP)** will be one of the following constant values:

L_SELECT—The <ENTER> key was pressed while the record was current.

S_CURRENT—The record just received focus because the user moved to it from another record or object.

S_NON_CURRENT—The record just lost focus because the user moved to another record or object.

S_SET_DATA—The fields must have their data set. The user function should use the data pointed to by *event.data* and the record number contained in *event.rawCode* to set the data in the fields

- $woFlags_{in}$ are flags (common to all window objects) that determine the general operation of the window. The following flags (declared in **UI_WIN.HPP)** affect the operation of a UIW_TABLE_RECORD class object:

WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag should not be used if the window has a UIW_BORDER object.

WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is 1/1 Oth the size of a normal cell by default.

WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags. This flag is set by default in the constructor.

UIW_TABLE_RECORD::DrawItem

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual advanced function is used to draw the object. If the WOS_OWNERDRAW status is set for the object, this function will be called when drawing the table record. This allows the programmer to derive a new class from UIW_TABLE_RECORD and handle the drawing of the table record, if desired. For the UIW_TABLE_RECORD object, the WOS_OWNERDRAW status is always set.

- $returnValue_{out}$ is a response based on the success of the function call. If the object is drawn the function returns a non-zero value. If the object is not drawn, 0 is returned.

- $event_{in}$ contains the run-time message that caused the object to be redrawn. *event.region* contains the region in need of updating. The following logical events may be sent to the **DrawItem(**) function:

  S_CURRENT, S_NON_CURRENT, S_DISPLAY_ACTIVE and **S_DISPLAY_INACTIVE**—Messages that cause the object to be redrawn.

  WM_DRAWITEM—A message that causes the object to be redrawn. This message is specific to Windows and OS/2.

Expose—A message that causes the object to be redrawn. This message is specific to Motif.

- *ccode*$_{in}$ contains the logical interpretation of *event.*

UIW_TABLE_RECORD::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the table record object. It is declared virtual so that any derived table record class can override its default operation.

- *returnValue*$_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the table record object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event():**

  L_SELECT—Indicates that the object has been selected.

  S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_SET_DATA—Causes the record to update the data in its fields. *event.rawCode* contains the record number and *event.data* contains the data for the record. If the UIW_TABLE_RECORD processes this message (i.e., the table record is not a derived table record) it will call the user function, if one exists, with a ccode of S_SET_DATA. *event* is sent to the user function.

All other events are passed by **Event()** to **UIW_WINDOW::Event( )** for processing.

UIW_TABLE_RECORD::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request*, void **data,*
ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the table record:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags,

particularly if the new flag settings will change the visual appearance of the object.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_GET_VALUE—Requests a pointer to the data associated with the record. If this request is sent, *data* should be a doubly-indirected pointer to void.

All other requests are passed by **Information()** to **UIW_WINDOW::Information()** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

UIW_TABLE_RECORD::RegionMax

Syntax

#include <ui_win.hpp>

virtual void RegionMax(UI_WINDOW_OBJECT **object);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function calculates the space *object* will occupy within the window and sets *object*->trueRegion accordingly.

- *object*$_{in}$ is a pointer to the object that is requesting the maximum region of the window. Its trueRegion member will be modified with its actual position.

UIW_TABLE_RECORD::VirtualRecord

Syntax

```
#include <ui_win.hpp>

UIW_TABLE_RECORD  *VirtualRecord(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function creates a copy of the table record. It does this by storing the record and reading it back in. Thus it is required that ZIL_STORE and ZIL_LOAD be defined when building the OpenZinc libraries if the UIW_TABLE object is to be used.

- *returnValue*$_{out}$ is a pointer to the new table record.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_TABLE_RECORD::UIW_TABLE_RECORD

Syntax

```
#include <ui_win.hpp>

UIW_TABLE_RECORD(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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_TABLE_RECORD by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a table record is stored in a data file it is usually stored as part of a UIW_TABLE and will be loaded when the table is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL,

the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TABLE_RECORD::Load

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_TABLE_RECORD from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TABLE_RECORD::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TABLE_RECORD::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual function returns a pointer to the object's New() function.

- *returnValue*$_{out}$ is a pointer to the object's New() function.

UIW_TABLE_RECORD::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

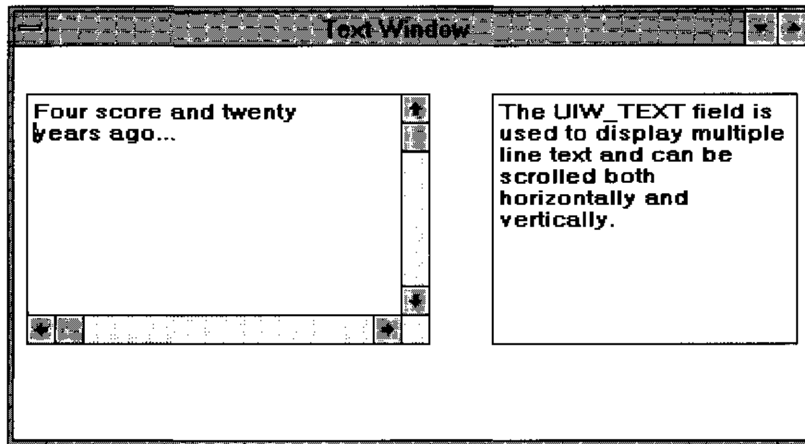
- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 28 - UIW TEXT

The UIW_TEXT class is used to display multiple-line text information and to collect information, in text form, from an end-user. Scroll bars can be added to the text field to allow both horizontal and vertical scrolling with the mouse. The figure below shows graphical implementations of a UIW_TEXT class object:



The UIW_TEXT class is declared in **UI_WIN**.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_TEXT : public UIW_WINDOW
{
public:
    static ZIL_ICHAR _className[];
    int insertMode;

    UIW_TEXT(int left, int top, int width, int height, ZIL_ICHAR *text,
        int maxLength = -1, WNF_FLAGS wnFlags = WNF_NO_WRAP,
        WOF_FLAGS woFlags = WOF_BORDER | WOF_AUTO_CLEAR,
        ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION));
    virtual ~UIW_TEXT(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    ZIL_ICHAR *DataGet(void);
    void DataSet(ZIL_ICHAR *text, int maxLength = -1);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
```

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

protected:
    int maxLength;
    ZIL_ICHAR *text;

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

    int CursorOffset(int offset = -1) ;
    void GetCursorPos(UI_POSITION *position);
    void SetCursorPos(const UI_POSITION &position);
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_TEXT class, *_className* is "UIW_TEXT."

- *insertMode* indicates whether the text object is in insert or overstrike mode. If *insertMode* is TRUE, the text is in insert mode. Otherwise the text is in overstrike mode.

- *maxLength* is the maximum length of the text buffer. *maxLength* does not include the NULL terminator.

- *text* is the text that is displayed in the text field.

UIW_TEXT::UIW_TEXT

Syntax

#include <ui_win.hpp>

UIW_TEXT(int *left,* int *top,* int *width,* int *height,* ZIL_ICHAR **text,* int *maxLength* = -1,
WNF_FLAGS *wnFlags* = WNF_NO_WRAP,
WOF_FLAGS *woFlags* = WOF_BORDER | WOF_AUTO_CLEAR,
ZIL_USER_FUNCTION *userFunction* = ZIL_NULLF(ZIL_USER_FUNCTION));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_TEXT class object.

- $left_{in}$ and top_{in} is the starting position of the text field within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the text field. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $height_{in}$ is the height of the text field. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

  $text_{in}$ is the text that is shown in the text field. Unless the WOF_NO_ALLOCATE_-DATA flag is set, the text is copied into a buffer, allocated by the UIW_TEXT object, which is *maxLength* + 1 characters in length. If the WOF_NO_-ALLOCATE_DATA flag is set, *text* must be space, allocated by the programmer, that is not deleted until the text field is deleted.

- *maxLength*$_{in}$ is the maximum length of the text buffer, excluding the NULL terminator. The UIW_TEXT object will automatically allocate extra space for the NULL terminator. If *maxLength* is -1 (default value), the size of the text buffer allocated is the initial length of *text,* including the NULL terminator. Please be aware that setting a large *maxLength* value may result in a smaller buffer being allocated due to memory limitations. For example, MS-Windows allocates the memory it uses for the text field from the local heap, so large buffers may not be allocated.

- *wnFlags*$_{in}$ are flags that define the operation of the text field. The following flags (declared in **UI_WIN.HPP)** affect the operation of a UIW_TEXT class object:

  WNF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WNF_-FLAGS.

  WNF_NO_WRAP—Prevents the text object from wrapping a line of text that is too wide to display on a single line. If this flag is not set, the text would automatically wrap to the next line. This flag is set by default in the constructor.

- *woFlags*$_{in}$ are flags (common to all window objects) that determine the general operation of the text object. The following flags (declared in **UI_WIN.HPP)** control the general presentation of, and interaction with, a UIW_TEXT class object:

  WOF_AUTO_CLEAR—Automatically marks the entire buffer if the end-user tabs to the field from another object. If the user then enters data (without first having pressed any movement or editing keys) the entire field will be replaced. This flag is set by default in the constructor.

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is l/10th the size of a normal cell by default.

WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor edit the text information. Typically, the field will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

WOF_UNANSWERED—Sets the initial status of the field to be "unanswered." An unanswered field is displayed as an empty field.

WOF_VIEW_ONLY—Prevents the object from being edited. However, the object may become current and the user may scroll through the data, mark it, and copy it.

- *userFunction*$_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. The user function will be called by the library when:

  1—the user moves onto the field, or

  2—the user moves to a different field in the window or to a different window.

  Because the user function is called at these times, the programmer can do data validation or any other type of necessary operation. The definition of the *userFunction* is as follows:

  EVENT_TYPE FunctionName(UI_WINDOW_OBJECT **object,*
  UI_EVENT *&event*, EVENT_TYPE *ccode)-*

  returnValue$_{out}$ indicates if an error has occurred. *returnValue* should be 0 if the no error occurred. Otherwise, the programmer should call the error system with an appropriate error message and return -1.

$object_{in}$ is a pointer to the object for which the user function is being called. This argument must be typecast by the programmer if class-specific members need to be accessed.

$event_{in}$ is the run-time message passed to the object.

$ccode_{in}$ is the logical or system code that caused the user function to be called. This code (declared in UI_WIN.HPP) will be one of the following constant values:

S_CURRENT—The object just received focus because the user moved to it from another field or window. This code is sent before any editing operations are permitted.

S_NON_CURRENT—The object just lost focus because the user moved to another field or window.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Add a text field to the window.
    UIW_WINDOW *window =
        UIW_WINDOW::Generic(0, 0, 40, 10, "Hello World Window");
    *window
        + new UIW_TEXT(0, 0, 0, 0, "Hello, World!", 1024,
            WNF_NO_WRAP, WOF_NON_FIELD_REGION);
    *windowManager + window;



    // The text object will automatically be destroyed when the window
    // is destroyed.
}
```

UIW_TEXT::~UIW_TEXT

Syntax

#include <ui_win.hpp>

virtual ~UIW_TEXT(void);

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_TEXT object.

UIW_TEXT::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This virtual function returns the object's class name.

- $returnValue_{out}$ is a pointer to *_className.*

UIW_TEXT::CursorOffset

Syntax

#include <ui_win.hpp>

int CursorOffset(int *offset* = -1);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function is used to get or set the edit cursor offset within the text field buffer.

- $returnValue_{out}$ is the character offset of the cursor within the buffer. For example, if the edit cursor is currently at character 137, *returnValue* will be 137.

- $offset_{in}$ indicates the character position at which the cursor should be placed. If *offset* is -1 (the default) the current cursor position is returned. The text field will be scrolled, if necessary, so that the new cursor location is in view.

UIW_TEXT::DataGet

Syntax

#include <ui_win.hpp>

ZIL_ICHAR *DataGet(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function returns the text information associated with the text object.

- *returnValue*$_{out}$ is a pointer to the text information associated with the text.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_TEXT *textl, UIW_TEXT *text2)
{
    ZIL_ICHAR *textData = textl->DataGet();
    text2->DataSet(textData);
}
```

UIW_TEXT::DataSet

Syntax

#include <ui_win.hpp>

void DataSet(ZIL_ICHAR **text*, int *maxLength* = -1);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function assigns new text to the UIW_TEXT object and redisplays the field. If no text is passed in (i.e., *text* is NULL), the field will be redrawn.

- $text_{in}$ is a pointer to the new text. If the WOF_NO_ALLOCATE_DATA flag is set, this argument must be text, allocated by the programmer, that is not destroyed until the UIW_TEXT class object is destroyed. Otherwise, the information associated with this argument is copied by the UIW_TEXT class object. If this argument is NULL, no text information is changed, but the text field is redisplayed.

- $maxLength_{in}$ is the number of characters to allocate for the text buffer. If *maxLength* is greater than the text's previous length, a new buffer of size *maxLength* + 1 (for the NULL terminator) is allocated. Otherwise, if *maxLength* is -1 or less than the size of the previous text, the previous buffer is used.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_TEXT *text1, UIW_TEXT *text2)
{
    ZIL_ICHAR *textData = text1->DataGet();
    text2->DataSet(textData);
}
```

UIW_TEXT::DrawItem

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual advanced function is used to draw the object. If the WOS_OWNERDRAW status is set for the object, this function will be called when drawing the text. This allows the programmer to derive a new class from UIW_TEXT and handle the drawing of the text, if desired.

- *returnValue*$_{out}$ is a response based on the success of the function call. If the object is drawn the function returns a non-zero value. If the object is not drawn, 0 is returned.

- *event*$_{in}$ contains the run-time message that caused the object to be redrawn. *event.region* contains the region in need of updating. The following logical events may be sent to the **DrawItem()** function:

  S_CURRENT, S_NON_CURRENT, S_DISPLAY_ACTIVE and **S_DISPLAY_INACTIVE**—Messages that cause the object to be redrawn.

  WM_DRAWITEM—A message that causes the object to be redrawn. This message is specific to Windows and OS/2.

  Expose—A message that causes the object to be redrawn. This message is specific to Motif.

- *ccode*$_{in}$ contains the logical interpretation of *event.*

UIW_TEXT::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function processes run-time messages sent to the text object. It is declared virtual so that any derived text class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the text object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event():**

  E_KEY—Indicates that a key has been pressed. It places the character in the text at the current cursor position unless the text is already *maxLength* characters. This message is interpreted from a keyboard event.

  L_BACKSPACE—Causes the first editable character to the left of the cursor position to be deleted and moves the cursor to that position. This message is interpreted from a keyboard event.

  L_BEGIN_MARK, L_CONTINUE_MARK and **L_END_MARK—**Communicate the progress of a mark operation. L_BEGIN_MARK indicates the marking process is beginning, L_CONTINUE_MARK indicates the growth or decrease of the marked region, and L_END_MARK indicates the end of the marking operation. Using a mouse, for example, L_BEGIN_MARK indicates that the mouse button has been pressed, L_CONTINUE_MARK indicates that the mouse is currently being dragged with the button depressed, and L_END_-MARK indicates that the mouse button has been released. These messages are interpreted from mouse events.

  L_BOL—Causes the cursor to move to the beginning of the current line. For example, where the underscore represents the cursor position, the text Stand and be counted would change to Stand and be counted as a result of the L_BOL message. If the mark feature is on, L_BOL extends the marked region to the beginning of the line. This message is interpreted from a keyboard event.

  L_BOTTOM—Scrolls the text to the last page and places the cursor at the end of the buffer. This message is interpreted from a keyboard event.

  L_COPY_MARK—Causes the marked region to be copied into the global paste buffer. This message is interpreted from a keyboard event.

L_CUT—Cuts the marked portion of the text. The cut region is stored in the global paste buffer. This message is interpreted from a keyboard event.

L_DELETE—Causes the marked characters, if any, or the character at the current cursor position to be deleted. For example, where the underscore represents the cursor position, the text Stand and be counted would change to Stand ad be counted as a result of the L_DELETE message. This message is interpreted from a keyboard event.

L_DELETE_EOL—Causes all characters from the current cursor position to the end of the line to be deleted. For example, where the underscore represents the cursor position, the text Stand and be counted would change to Stand a_ as a result of the L_DELETE_EOL message. This message is interpreted from a keyboard event.

L_DELETE_WORD—Causes the word at the cursor position to be deleted, along with any trailing spaces. For example, where the underscore represents the cursor position, the text Stand and be counted would change to Stand be counted as a result of the L_DELETE_WORD message. This message is interpreted from a keyboard event.

L_DOWN—Causes the cursor to move down one line in the text buffer. Where possible, the cursor position stays at the same horizontal character offset. This message is interpreted from a keyboard event.

L_EOL—Causes the cursor to move to the end of the current line. For example, where the underscore represents the cursor position, the text Stand and be counted would change to Stand and be counted_ as a result of the L_EOL message. If the mark feature is on, L_EOL extends the marked region to the end of the line. This message is interpreted from a keyboard event.

LJLEFT—Causes the cursor to move one character or space to the left of its current position, if it is not in the field's first editable position. If the mark feature is on, L_LEFT extends the marked region to include the character. This message is interpreted from a keyboard event.

L_MARK_BOL—Marks the text from the current cursor position to the beginning of the current line and places the cursor at the beginning of the line. This message is interpreted from a keyboard event.

L_MARK_DOWN—Causes the cursor to move down one line in the text buffer. Where possible, the cursor position stays at the same horizontal character offset.

The text between the starting cursor position and the ending cursor position will be marked. This message is interpreted from a keyboard event.

L_MARK_EOL—Marks the text from the current cursor position to the end of the current line and places the cursor at the end of the line. This message is interpreted from a keyboard event.

L_MARK_LEFT—Moves the cursor to the left one character, marking the character. This message is interpreted from a keyboard event.

L_MARK_PGDN—Causes the text field to scroll down one page. The text between the starting cursor position and the ending cursor position will be marked. This message is interpreted from a keyboard event.

L_MARK_PGUP—Causes the text field to scroll up one page. The text between the starting cursor position and the ending cursor position will be marked. This message is interpreted from a keyboard event.

L_MARK_RIGHT—Moves the cursor to the right one character, marking the character. This message is interpreted from a keyboard event.

L_MARK_UP—Causes the cursor to move up one line in the text buffer. Where possible, the cursor position stays at the same horizontal character offset. The text between the starting cursor position and the ending cursor position will be marked. This message is interpreted from a keyboard event.

L_MARK_WORD_LEFT—Causes the cursor position to be moved to the beginning of the current word or, if the cursor is at the beginning of the current word, to the beginning of the next word to the left of the current cursor position. The text between the starting cursor position and the ending cursor position will be marked. This message is interpreted from a keyboard event.

L_MARK_WORD_RIGHT—Causes the cursor to move to the beginning of the next word to the right of the current cursor position. The text between the starting cursor position and the ending cursor position will be marked. This message is interpreted from a keyboard event.

L_PASTE—Causes the contents of the paste buffer to be placed in the field at the current cursor position. For example, if the contents of the paste buffer were up, the text Stand and be counted would change to Stand up and be counted. If the contents are too large for the text field, only that portion which fits will be pasted. This message is interpreted from a keyboard event.

L_PGDN—Causes the text field to scroll down one page. This message is interpreted from a keyboard event.

L_PGUP—Causes the text field to scroll up one page. This message is interpreted from a keyboard event.

L_RIGHT—Causes the cursor to move one character or space to the right of its current position, if it is not in the field's last editable position. If the mark feature is on, L_RIGHT extends the marked region to include the character. This message is interpreted from a keyboard event.

L_SELECT—Indicates that the object has been selected. The selection may be the result of a mouse click or a keyboard action.

L_TOP—Scrolls the text to the first page and places the cursor at the beginning of the buffer. This message is interpreted from a keyboard event.

L_UP—Causes the cursor to move up one line in the text buffer. Where possible, the cursor position stays at the same horizontal character offset. This message is interpreted from a keyboard event.

L_VIEW—Indicates that the mouse is being moved over the text field. This message allows the field to alter the mouse image.

L_WORD_LEFT—Causes the cursor position to be moved to the beginning of the current word or, if the cursor is at the beginning of the current word, to the beginning of the next word to the left of the current cursor position. For example, where the underscore represents the cursor position, the text Stand and be counted would change to Stand and be counted, as a result of the L_WORD_LEFT message. This message is interpreted from a keyboard event.

L_WORD_RIGHT—Causes the cursor to move to the beginning of the next word to the right of the current cursor position. For example, where the underscore represents the cursor position, the text Stand and be counted would change to Stand and be counted as a result of the L_WORD_RIGHT message. This message is interpreted from a keyboard event.

S_ADD_OBJECT—Causes a new object to be added to the text field, such as a scroll bar. *event.data* will point to the new object to be added.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate

their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_CURRENT—Causes the object to draw itself to appear current. This message is sent by the Window Manager to a window when it becomes current. The window, in turn, passes this message to the object on the window that is current.

S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager sends this message to a window when the window is subtracted from the Window Manager. The window, in turn, relays the message to all objects attached to it.

S_DISPLAY_ACTIYE—Causes the object to draw itself to appear active. An active object is one that is on the active (i.e., current) window. Most objects do not display differently whether they are active or inactive. An active object should not be confused with a current object. An object is active if it is on the active window. However, it may not be the current object on the window.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed by the event overlaps the region of the object. This message is sent by a UIW_WINDOW to all the non-current, active objects attached to it.

S_DISPLAY_INACTIVE—Causes the object to draw itself to appear inactive. An inactive object is one that is not on the active (i.e., current) window. Most objects do not display differently whether they are inactive or active.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed with the event overlaps the region of the object. This message is sent by a UIW_WINDOW to all the objects attached to it.

S_HSCROLL—Causes the text to scroll its contents horizontally, *event.scroll-delta* indicates how far to scroll.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_MOVE—Causes the object to update its location. The distance to move is contained in the *position* field of UI_EVENT. For example, an *event.position.-line* of -10 and an *event.position.column* of 15 moves the object 10 lines up and 15 columns to the right.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another field or window.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

S_SUBTRACT_OBJECT—Causes an object to be subtracted from the list, such as a scroll bar. *event.data* will point to the object to be subtracted.

S_VERIFY_STATUS—Causes the object to correlate its state (e.g., cursor position) with the operating system.

S_VSCROLL—Causes the text to scroll vertically, *event.scroll.delta* indicates how far to scroll.

All other events are passed by Event() to UIW_WINDOW::Event() for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived Event( ) function.

UIW_TEXT::GetCursorPos

Syntax

```
#include <ui_win.hpp>

void GetCursorPos(UI_POSITION *position);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function returns the current cursor xy-position within the visible text field.

- *$position_{out}$* is a pointer to the UI_POSITION structure to which the cursor location information will be copied. The cursor location is relative to the top-left corner of the field, *position.column* will contain the character offset from the left edge of the field and *position.line* will contain the character offset from the top edge of the field.

UIW_TEXT::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request,* void **data,*
ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *$returnValue_{out}$* is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in UI_WIN.HPP) are recognized by the text object:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_COPY_TEXT—Copies the *text* associated with the object into a buffer provided by the programmer. If this request is sent, *data* must be the address of a buffer where the text will be copied. This buffer must be large enough to contain all of the characters associated with the text and the terminating NULL character.

  I_GET_MAXLENGTH—Gets the *maxLength* value. If *data* is NULL, the address of *maxLength* will be returned. Otherwise, *data* should be a pointer to an integer where *maxLength* will be copied.

  I_GET_TEXT—Returns a pointer to the text associated with the object. If this request is sent, *data* should be a doubly-indirected pointer to ZIL_ICHAR. This request does not copy the text into a new buffer.

  I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

  I_SET_MAXLENGTH—Sets the *maxLength* member. If this request is sent *data* should be a pointer to an integer that contains the new maximum length.

  I_SET_TEXT—Sets the text associated with the object. This request will also redisplay the object with the new text, *data* should be a pointer to the new text.

  All other requests are passed by **Information()** to **UIW_WINDOW::Information()** for processing.

- *data*$_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- *objectID*$_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information( )** function is virtual, it is possible for an

object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include <ui_win.hpp>

ExampleFunction()
{
    ZIL_ICHAR string[30];
    text-information(I_COPY_TEXT, string);

    text1->Information(I_SET_TEXT, "First name:");
    text2->Information(I_SET_TEXT, "Last name:");
```

UIW_TEXT::SetCursorPos

Syntax

#include <ui_win.hpp>

void SetCursorPos(const UI_POSITION *&position);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function is used to set the cursor position within the displayed text field.

- $position_{in}$ is a pointer to the UI_POSITION structure that is to contain the new cursor position, *position.column* is the character offset from the left edge of the field, and *position.line* is the character offset from the top edge of the field.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_TEXT::UIW_TEXT

Syntax

```
#include <ui_win.hpp>

UIW_TEXT(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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_TEXT by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a text object is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter

69—ZIL_STORAGE_OBJECT_READ_ONLY" *of Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UIJWINDOWJOB JECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TEXT::Load

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_TEXT from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW TEXT::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" *of Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW TEXT::NewFunction

Syntax

#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's New() function.

- *returnValue*$_{out}$ is a pointer to the object's New( ) function.

UIW_TEXT::Store_

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- $name_{in}$ is the name of the object to be stored.

- $file_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

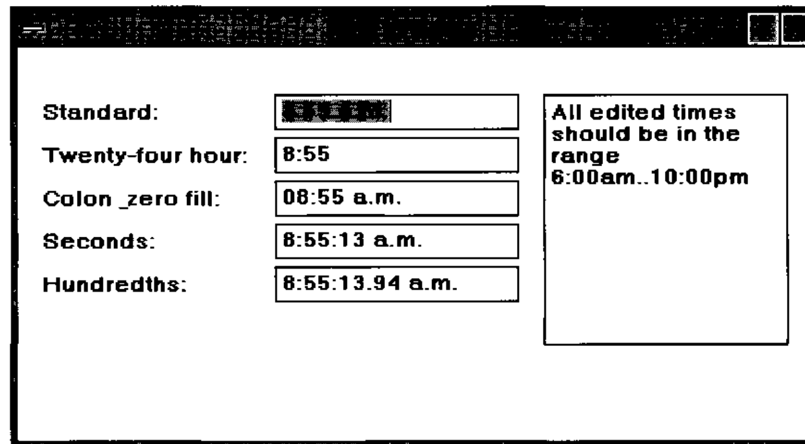
- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 29 - UIW_TIME

The UIW_TIME class is used to display time information to the screen and to collect information, in time format, from an end-user. The UIW_TIME class will automatically format the displayed time. The UIW_TIME class is a high-level object that is used to interact with the end-user. It makes use of the ZIL_TIME class, which is a low-level object that handles the details of time data manipulation. See "Chapter 72—ZIL_TIME" of *Programmer's Reference Volume 1* for more information about the ZIL_TIME class. The figure below shows graphical implementations of UIW_TIME objects:



The UIW_TIME class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_TIME : public UIW_STRING
{
public:
    static ZIL_ICHAR _className[];
    static int defaultInitialized;
    TMF_FLAGS tmFlags;
#if defined(ZIL_3x_COMPAT)
    static TMF_FLAGS rangeFlags;
#endif

    UIW_TIME(int left, int top, int width, ZIL_TIME *time,
        const ZIL_ICHAR *range = ZIL_NULLP(ZIL_ICHAR),
        TMF_FLAGS tmFlags = TMF_NO_FLAGS,
        WOF_FLAGS woFlags = WOF_BORDER | WOF_AUTO_CLEAR,
        ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION));
    virtual ~UIW_TIME(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    ZIL_TIME *DataGet(void);
    void DataSet(ZIL_TIME *time);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);
```

```
    virtual int Validate(int processError = TRUE);

#if defined (ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_TIME(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
#if defined (ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

    void SetLanguage(const ZIL_ICHAR *languageName);

protected:
    ZIL_TIME *time;
    ZIL_ICHAR *range;
    const ZIL_LANGUAGE *myLanguage;
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_TIME class, *_className* is "UIW_TIME."

- *defaultlnitialized* indicates if the default language strings for this object have been set up. The default strings are located in the file LANG_DEF.CPP. If *defaultlnitialized* is TRUE, the strings have been set up. Otherwise they have not been.

- *rangeFlags* are flags that define how the range values are interpreted. *rangeFlags* is set to TMF_SECONDS | TMFJHUNDREDTHS by default.

- *tmFlags* are flags that define the operation of the UIW_TIME class. A full description of the time flags is given in the UIW_TIME constructor.

- *time* is a pointer to a ZIL_TIME that is used to manage the low-level time information. If the WOF_NO_ALLOCATE_DATA flag is set, this member will simply point to the ZIL_TIME value passed in.

- *range* is a string that specifies the range(s) of acceptable time values, *range* is a copy of the range that is passed to the constructor.

- *myLanguage* is the ZIL_LANGUAGE object that contains the string translations for this object.

UIW_TIME::UIW_TIME

Syntax

```
#include <ui_win.hpp>

UIW_TIME(int left, int top, int width, ZIL_TIME *time,
    const ZIL_ICHAR *range = ZIL_NULLP(ZIL_ICHAR),
    TMFJLAGS tmFlags = TMF_NO_FLAGS,
    WOF.FLAGS woFlags = W0FJ30RDER | WOF_AUTO_CLEAR,
    ZIL_USER_FUNCTION userFunction = ZIL_NULLF(ZIL_USER_FUNCTION));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_TIME class object.

- $left_{in}$ and top_{in} is the starting position of the time field within its parent window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the time field. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value. The height is determined automatically by the UIW_TIME object.

- *time*$_{in}$ is a pointer to a ZIL_TIME object. Its value will be used as the initial value.

- *range*$_{in}$ is a string that specifies the valid time ranges. A range consists of a minimum value, a maximum value, and the values in between. For example, if a range of "1:30..2:30" is specified, the UIW_TIME class object will only accept those time values that fall between one-thirty and two-thirty, inclusive. Open-ended ranges can be specified by leaving the minimum or maximum value off. For example, a range of "2:30.." will allow all times that are two-thirty or thereafter. Multiple, disjoint ranges can be specified by separating the individual ranges with a slash (i.e., '/'). For example, "l:30..2:30/3:30.." will accept all times between one-thirty and two-thirty and all times after three-thirty. If *range* is NULL, any time within the absolute range is accepted. This string is copied by the UIW_TIME class object to the *range* member variable.

- *tmFlags*$_{in}$ describes how the time should display and interpret the time information. The following flags (declared in UI_GEN.HPP) control the general presentation of a UIW_TIME class object:

Flag	Example
TMF_COLON_SEPARATOR—Formats the time with colons separating the time information.	12:00 13:00:00 12:00 a.m.
TMF_HUNDREDTHS—Formats the time with a hundredths of seconds value.	1:05:00.00 23:15:05.99 7:45:59.00 a.m.
TMF_LOWER_CASE—Converts the time to lower-case.	12:00 p.m. 1:00 a.m.
TMF_NO_FLAGS—Does not associate any special flags with the ZIL_TIME object. In this case, the time will be formatted using the default country information. This is the default argument if no other argument is specified. This flag should not be used in conjunction with any other TMF flags.	12:00 13:00:00 12:00 a.m.
TMF_NO_HOURS—Formats the time with no hour.	37:59 56:43.99
TMF_NO_MINUTES—Formats the time with no minute value.	12:56 11:45.99

TMF_NO_SEPARATOR—Does not place a separator between time information.

```
1200
130000
```

TMF_SECONDS—Formats the time with a seconds value.

```
12 : 00 : 05 a.m.
1:13:25
16:00:00
```

TMF_SYSTEM—Sets the time value according to the system time.

```
12:00 : 05 a.m.
1:13 :25
16:00:00
```

TMF_TWELVE_HOUR—Formats the time using a 12 hour format, regardless of the default country information.

```
12:00 a.m.
1:00 p.m.
5:00 p.m.
```

TMF_TWENTY_FOUR_HOUR—Formats the time using a 24 hour format, regardless of the default country information.

```
12 : 00
13 : 00
17:00
```

TMF_UPPER_CASE—Converts the time to upper-case.

```
12:00 P.M.
1:00 A.M.
```

TMF_ZERO_FILL—Forces the hour, minute and second values to be zero filled when their values are less than 10.

```
01:10 a.m
13:05:03
01:01 p.m.
```

- *woFlags*$_{in}$ are flags (common to all window objects) that determine the general operation of the time object. The following flags (declared in **UI_WIN.HPP)** control the general presentation of, and interaction with, a UIW_TIME class object:

  WOF_AUTO_CLEAR—Automatically marks the entire buffer if the end-user tabs to the field from another object. If the user then enters data (without first having pressed any movement or editing keys) the entire field will be replaced. This flag is set by default in the constructor.

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOF_INVALID—Sets the initial status of the field to be "invalid." Invalid entries fit in the absolute range determined by the object type but do not fulfill

all the requirements specified by the program. For example, a time may initially be set to 1:30, but the final time, edited by the end-user, must be in the range "6:30..7:30." The initial time in this example fits the absolute range requirements of a UIW_TIME class object but does not fit into the specified range. By denoting the field as invalid, you force the user to enter an acceptable value.

WOF_JUSTIFY_CENTER—Center-justifies the data within the displayed field.

WOF_JUSTIFY_RIGHT—Right-justifies the data within the displayed field.

WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is l/10th the size of a normal cell by default.

WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

WOF_NO_FLAGS—Does not associate any special window flags with the object. Setting this flag left-justifies the data. This flag should not be used in conjunction with any other WOF flags.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor edit the time information. Typically, the field will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

WOF_SUPPORT_OBJECT—Causes the object to be placed in the parent object's support list. The support list is reserved for objects that are not displayed as part of the user region of the window. Care should be used when setting this flag on an object that does not use it by default as undesirable effects may occur. This flag generally should not be used by the programmer.

WOF_UNANSWERED—Sets the initial status of the field to be "unanswered." An unanswered field is displayed as an empty field.

WOF_VIEW_ONLY—Prevents the object from being edited. However, the object may become current and the user may scroll through the data, mark it, and copy it.

- *userFunction*$_{in}$ is a programmer defined function that will be called by the library at certain points in the user's interaction with an object. The user function will be called by the library when:

  1—the user moves onto the field,

  2—the <ENTER> key is pressed while the field is current or, if the field is in a list, the mouse is clicked on it, or

  3—the user moves to a different field in the window or to a different window.

  Because the user function is called at these times, the programmer can do data validation or any other type of necessary operation. The definition of the *userFunction* is as follows:

  EVENT_TYPE FunctionName(UI_WINDOW_OBJECT **object,*
  UI_EVENT *&event,* EVENT_TYPE *ccode);*

  returnValue$_{out}$ indicates if an error has occurred. *returnValue* should be 0 if the no error occurred. Otherwise, the programmer should call the error system with an appropriate error message and return -1.

  object$_{in}$ is a pointer to the object for which the user function is being called. This argument must be typecast by the programmer if class-specific members need to be accessed.

  event$_{in}$ is the run-time message passed to the object.

  ccode$_{in}$ is the logical or system code that caused the user function to be called. This code (declared in **UI_WIN.HPP)** will be one of the following constant values:

  L_SELECT—The <ENTER> key was pressed while the field was current or, if the field is in a list, the mouse was clicked on the field.

  S_CURRENT—The object just received focus because the user moved to it from another field or window. This code is sent before any editing operations are permitted.

S_NON_CURRENT—The object just lost focus because the user moved to another field or window.

NOTE: If a user function is associated with the object, Validate() must be called explicitly from within *userFunction* if range checking is desired.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI__WINDOW_MANAGER *windowManager)
{
    ZIL_DATE date; // system date
    ZIL_TIME time; // system time

    // Create a window with a date and time field.
    UIW_WINDOW *window = UIW_WINDOW::Generic(0, 0, 45, 8, "Window");
    *window
        + new UIW_PROMPT(2, 1, "Date..")
        + new UIW_DATE(9, 1, 20, &date, ZIL_NULLP(ZIL_ICHAR),
            DTF_ALPHA_MONTH | DTF_SYSTEM)
        + new UIW_PROMPT(2, 3, "Time..")
        + new UIW_TIME(9, 3, 20, atime, ZIL_NULLP(ZIL_ICHAR), TMF_SECONDS);



    // The time object will automatically be destroyed when the window
    // is destroyed.
}
```

UIW_TI M E::~UIW_TIME

Syntax

#include <ui_win.hpp>

virtual ~UIW_TIME(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_TIME object.

UIW_TIME::ClassName

Syntax

#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the object's class name.

- $returnValue_{out}$ is a pointer to *_className.*

UIW_TIME::DataGet

Syntax

#include <ui_win.hpp>

ZIL_TIME *DataGet(void);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function gets the current time information associated with the UIW_TIME class object.

- *returnValue*$_{out}$ is a pointer to a ZIL_TIME object containing the current time value.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_TIME *timeObject)
{
    ZIL_TIME *time = timeObject->DataGet();


}
```

UIW_TIME;: DataSet

Syntax

#include <ui_win.hpp>

void DataSet(ZIL_TIME **time);*

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function assigns a new value to the UIW_TIME object and redisplays the field. If no value is passed in (i.e., *value* is NULL), the field will be redrawn.

- $value_{in}$ is a pointer to the new time. If the WOF_NO_ALLOCATE_DATA flag is set, this argument must be a ZIL_TIME, allocated by the programmer, that is not destroyed until the UIW_TIME class object is destroyed. Otherwise, the information associated with this argument is copied by the UIW_TIME class object. If this argument is NULL, no time information is changed, but the time field is redisplayed.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_TIME *time)
{
    ZIL_TIME timelnfo(12, 0, 30);
    time->DataSet(fctimelnfo);
}
```

UIW_TIME::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the time object. It is declared virtual so that any derived time class can override its default operation.

- *returnValue*$_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the time object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event():**

  L_SELECT—Indicates that the object has been selected. The selection may be the result of a mouse click or a keyboard action.

  S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

  S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another field or window.

  All other events are passed by **Event()** to **UIW_STRING::Event()** for processing.

UIW_TIME::Information

Syntax

```
#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST request, void *data,
    ZIL_OBJECTID objectID = ID_DEFAULT);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the time object:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_CLEAR_FLAGS—Clears the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be cleared, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to clear the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_-OBJECT. If the TMF_FLAGS are to be cleared, *objectID* should be ID_TIME. This allows the object to process the request at the proper level. This request only clears those flags that are passed in; it does not simply clear the entire field.

  I_GET_FLAGS—Requests the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to obtain the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the TMF_FLAGS are desired, *objectID* should be ID_TIME. This allows the object to process the request at the proper level.

  I_GET_VALUE—Gets the current value for the object. If this request is sent, *data* should be a pointer to ZIL_TIME. If *data* is NULL *returnValue* will return a pointer to ZIL_TIME.

  I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_FLAGS—Sets the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be set, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to set the WOF_-FLAGS of an object, *objectID* should be ID_WINDOW_OB JECT. If the TMF_-FLAGS are to be set, *objectID* should be ID_TIME. This allows the object to process the request at the proper level. This request only sets those flags that are passed in; it does not clear any flags that are already set.

I_SET_VALUE—Sets the current value for the object. If this request is sent, *data* should be a pointer to a ZIL_TIME that contains the value to be set.

All other requests are passed by **Information()** to **UIW_STRING::Information( )** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include <ui_win.hpp>

ExampleFunction()
{
    time->InfoCTiiation(I_SET_TEXT, "12:00");


    woFlags flags = WOF_BORDER;
    time->information(I_SET_FLAGS, &flags);


}
```

UIW_TIME::SetLanguage

Syntax

```
#include <ui_win.hpp>

void SetLanguage(const ZIL_ICHAR *languageName);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function sets the language to be used by the object. The string translations for the object will be loaded and the object's *myLanguage* member will be updated to point to the new ZIL_LANGUAGE object. By default, the object uses the language identified in the **LANG_DEF.CPP** file, which compiles into the library. (If a different default language is desired, simply copy a **LANG_<ISO>.CPP** file from the OpenZinc\SOURCE\-INTL directory to the \OpenZinc\SOURCE directory, and rename it to **LANG_DEF.CPP** before compiling the library.) The language translations are loaded from the **I18N.DAT** file, so it must be shipped with your application.

- *languageName*$_{in}$ is the two-letter ISO language code identifying which language the object should use.

UIW_TIME::Validate

Syntax

```
#include <ui_win.hpp>

virtual int Validate(int processError = TRUE);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function is used to validate objects. When an object receives the S_CURRENT or S_NON_CURRENT messages, it calls **Validate()** to check if the value entered is valid. However, if a user function is associated with the object, **Validate()** must be called explicitly from the user function if range checking is desired. The value is invalid if it is not within the absolute range of the object or if it is not within a range specified by the *range* member variable.

- *returnValue*$_{out}$ indicates the result of the validation. The possible values for *returnValue* are:

  TMI_GREATER_THAN_RANGE—The time was greater than the maximum value of a negatively open-ended range.

  TMI_INVALID—The time was invalid or was in an invalid format.

  TMI_LESS_THAN_RANGE—The time was less than the minimum value of a positively open-ended range.

  TMI_OK—The time was successfully imported.

  TMI_OUT_OF_RANGE—The time was out of the valid range for times.

  TMI_VALUE_MISSING—All of the required field values were not present.

- *processError*$_{in}$ determines whether **Validate()** should call **UIJERRORJSYSTEM::-ReportError()** if an error occurs. If *processError* is TRUE, **ReportError()** is called. Otherwise, the error system is not called.

Example

```
#include <ui_win.hpp>

EVENT_TYPE TimeUserFunction(UI_WINDOW_OBJECT *object, UI_EVENT &,
    EVENT_TYPE ccode)
```

```
{
    if (ccode != S_NON_CURRENT)
        return (ccode);

    // Do specific validation.
    ZIL_TIME currentTime;
    ZIL_TIME *time = ((UIW_TIME *)object)->DataGet();

    // Call the default Validate function to check for valid time,
    int valid = object->Validate(TRUE);

    // Call error system if the time entered is later than the system time.
    if (valid == TMI_OK && currentTime < *time)
    {
        valid = TMI_INVALID;
        ZIL_ICHAR timeString[64];
        currentTime.Export(timeString, TMF_NO_FLAGS);
        object->errorSystem->ReportError(object->windowManager,WOS_NO_STATUS,
            "The time must be before %s.", timeString);
    }
    // Return error status.
    if (valid == TMI_OK)
        return (0);
    else
        return (-1);

void ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    UIW_WINDOW *window = UIW_WINDOW::Generic(0, 0, 45, 8, "VCR Control");
    *window
        + new UIW_PROMPT(2, 1, "Start time:")
        + new UIW_TIME(12, 1, 20, &ZIL_TIME(), NULL, TMF_NO_FLAGS,
            WOF_BORDER | WOF_AUTO_CLEAR, TimeUserFunction)
        + new UIW_PROMPT(2, 3, "End time:")
        + new UIW_TIME(12, 3, 20, &ZIL_TIME(), NULL, TMF_NO_FLAGS,
            WOF_BORDER | WOF_AUTO_CLEAR, TimeUserFunction);
    *windowManager + window;
```

Storage Members

This section describes those class members that are used for storage purposes.

UIW_TIME::UIW_TIME

Syntax

#include <ui_win.hpp>

UIW_TIME(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));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_TIME by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a time object is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" *of Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume* |. If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TIME::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_TIME from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *$userTable_{in}$* is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TIME::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- *$name_{in}$* is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UIJWIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UIJWINDOWJOB JECT: .userTable* in "Chapter 43—UI_WINDOW.-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TIME::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New( )** function.

- *returnValue*$_{out}$ is a pointer to the object's **New()** function.

UIW_TIME::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- *name*$_{in}$ is the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

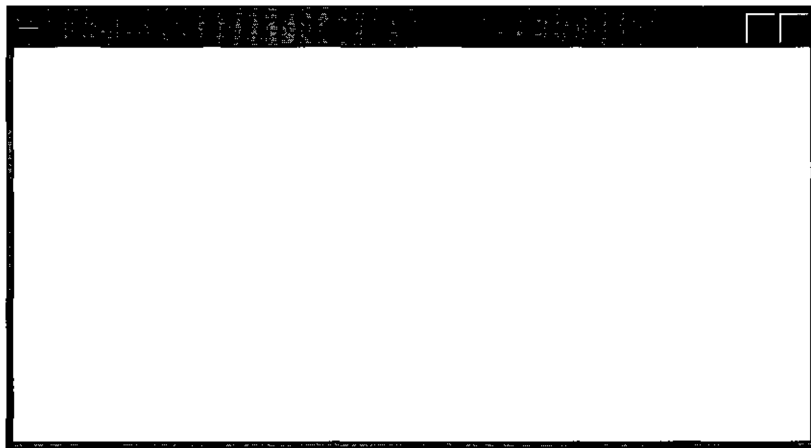
- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see

the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 30 - UIW_TITLE

The UIW_TITLE class is used to display the title of a window or to allow movement of the window with the mouse. The window can also be maximized by double-clicking on the title with the mouse, or if the window is already maximized, the window can be restored to its original size by double-clicking on it. The figure below shows a graphical implementation of a window with a UIW_TITLE class object (the bar with the "Generic Window" text):



NOTE: The Macintosh does not have the concept of maximizing or restoring windows. Because of this, double-clicking on a title bar will not perform these tasks in a Macintosh application built with OpenZinc. OpenZinc will ignore the double-click and no action will be performed.

The UIW_TITLE class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_TITLE : public UIW_BUTTON
{
public:
    static ZIL_ICHAR _className[];

    UIW_TITLE(ZIL_ICHAR    *text,    WOF_FLAGS   woFlags   =   WOF_BORDER |
        WOF_JUSTIFY_CENTER);
    virtual ~UIW_TITLE(void);
    virtual ZIL_ICHAR *ClassName(void);
    ZIL_ICHAR *DataGet(void);
    void DataSet(ZIL_ICHAR *text);
    virtual EVENT_TYPE Event (const UI_EVENT Seventh-
    virtual void information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);
```

```
#if defined (ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_TITLE(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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif
};
```

General Members

This section describes those members that are used for general purposes.

_className contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_TITLE class, *_className* is "UIW_TITLE."

UIW_TITLE::UIW_TITLE

Syntax

#include <ui_win.hpp>

UIW_TITLE(ZIL_ICHAR *"text,* WOF_FLAGS *woFlags* = WOF_BORDER | WOF_JUSTIFY_CENTER);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_TITLE class object.

- $text_{in}$ is the text to display in the title bar.

- $woFlags_{in}$ are flags (common to all window objects) that define the operation of the UIW_TITLE class. The following flags (declared in **UI_WIN.HPP)** affect the operation of the UIW_TITLE class object:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

  WOF_JUSTIFY_CENTER—Center-justifies the data within the displayed field. This flag is set by default in the constructor. This flag may have no effect in some operating systems (e.g., OS/2) as the operating system controls where the text appears in the title bar.

  WOF_JUSTIFY_RIGHT—Right-justifies the data within the displayed field. This flag may have no effect in some operating systems (e.g., OS/2) as the operating system controls where the text appears in the title bar.

  WOF_NO_ALLOCATE_DATA—Prevents the object from allocating a buffer to store the data. If this flag is set, the programmer is responsible for allocating the memory for the data. The programmer is also responsible for deallocating that memory when it is no longer needed.

  WOF_NO_FLAGS—Does not associate any special window flags with the object. Setting this flag left-justifies the data. This flag should not be used in conjunction with any other WOF flags.

The title bar object is always positioned along the top edge of the parent window between the system button, if one exists, and the maximize and minimize buttons, if they exist. To ensure that the title bar is drawn correctly, it must be added after the maximize button, minimize button and system button have been added. The following example shows the correct order of title bar addition.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Create a basic window.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10)
    *window
        + new UIW_BORDER
        + new UIW_MAXIMIZE_BUTTON
        + new UIW_MINIMIZE_BUTTON
        + new UIW_SYSTEM_BUTTON
        + new UIW_TITLE("Window 1");
    *windowManager + window;



    // The title object will automatically be destroyed when the window
    // is destroyed.
}
```

UIW_TITLE::~UIW_TITLE

Syntax

#include <ui_win.hpp>

virtual ~UIW_TITLE(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_TITLE object.

UIW TITLE::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This virtual function returns the object's class name.

- $returnValue_{out}$ is a pointer to _className.

UIW TITLE::DataGet

Syntax

```
#include <ui_win.hpp>

ZIL_ICHAR        *DataGet(void);
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function is used to return the text information associated with the title object.

- *returnValue*$_{out}$ is a pointer to the text information associated with the title.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_TITLE *title)
{
    ZIL_ICHAR *titleText = title->DataGet();


}
```

UIW_TITLE::DataSet

Syntax

#include <ui_win.hpp>

void DataSet(ZIL_ICHAR **text);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function is used to set the text information associated with the title.

- *text*$_{in}$ is a pointer to the new text information to be displayed on the title bar.

Example

```
#include <ui_win.hpp>

ExampleFunction(UIW_TITLE *title)
{
```

```
    title->DataSet("Error Window");
}
```

UIW_TITLE;: Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function processes run-time messages sent to the title bar object. It is declared virtual so that any derived title bar class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the title bar object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by Event():

  L_BEGIN_SELECT—Indicates that the end-user began the selection of the object by pressing the mouse button down while on the object.

  S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

  S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system.

This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_DISPLAY_ACTIVE—Causes the object to draw itself to appear active. An active object is one that is on the active (i.e., current) window. Most objects do not display differently whether they are active or inactive. The UIW_TITLE and UIW_BORDER class objects usually do draw differently if they are active than if they are inactive. This is done to allow the end-user to easily distinguish the current window from the rest of the windows. An active object should not be confused with a current object. An object is active if it is on the active window. However, it may not be the current object on the window.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed by the event overlaps the region of the object. This message is sent by a UIW_WINDOW to all the non-current, active objects attached to it.

S_DISPLAY_INACTIVE—Causes the object to draw itself to appear inactive. An inactive object is one that is not on the active (i.e., current) window. Most objects do not display differently whether they are inactive or active. The UIW_TITLE and UIW_BORDER class objects usually do draw differently if they are inactive than if they are active. This is done to allow the end-user to easily distinguish the current window from the rest of the windows.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed with the event overlaps the region of the object. This message is sent by a UIW_WINDOW to all the objects attached to it.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_REDISPLAY—Causes the object to redraw.

All other events are passed by **Event()** to **UIW_BUTTON::Event()** for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own

messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived **Event()** function.

UIW_TITLE::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request,* void *"data,*
ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- $returnValue_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- $request_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the Window Manager:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_COPY_TEXT—Copies the *text* associated with the object into a buffer provided by the programmer. If this request is sent, *data* must be the address of

a buffer where the string's text will be copied. This buffer must be large enough to contain all of the characters associated with the title and the terminating NULL character.

I_GET_TEXT—Returns a pointer to the text associated with the object. If this request is sent, *data* should be a doubly-indirected pointer to **ZIL_ICHAR.** This request does not copy the text into a new buffer.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_TEXT—Sets the text associated with the object. This request will also redisplay the object with the new text, *data* should be a pointer to the new text. Also, the **Information( )** function of the window to which the title bar is attached can be called with the I_SET_TEXT request. This request will be sent to the title. This will eliminate the necessity to obtain a pointer to the title bar.

All other requests are passed by **Information( )** to **UIW_BUTTON::Information( )** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include <ui_win.hpp>
ExampleFunction()
{
    title-information(I_SET_TEXT, "Example 2")
```

Storage Members

This section describes those class members that are used for storage purposes.

UIW_TITLE::UIW_TITLE

Syntax

#include <ui_win.hpp>

UIW_TITLE(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));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIWJTTLE by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a title bar is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter

69—ZIL_STORAGE_OBJECT_READ_ONLY" *of Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TITLE::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_TITLE from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TITLE::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:-.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TITLE::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's New() function.

- $returnValue_{out}$ is a pointer to the object's New( ) function.

UIW_TITLE::Store

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- $name_{in}$ is the name of the object to be stored.

- $file_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

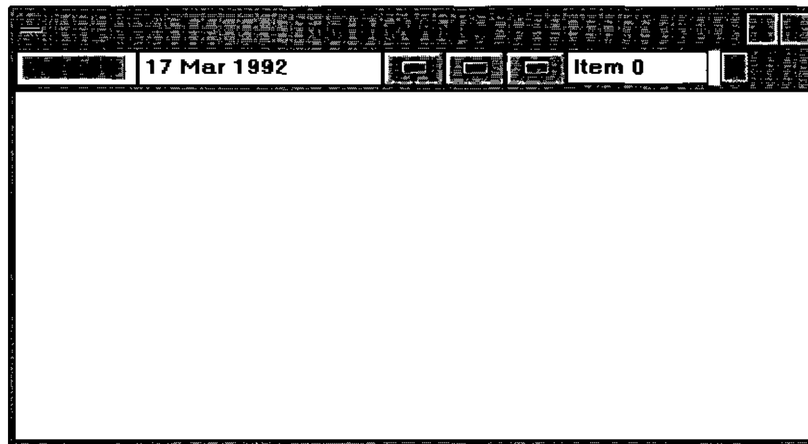
- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 31 - UIW_TOOL_BAR

The UIW_TOOL_BAR class object is used as a controlling structure for a set of window objects. Typically, a tool bar is used to provide a method for quickly selecting an option or action that is frequently chosen in an application. Objects on a tool bar may be editable, if desired. A tool bar is positioned along the top of the window. If the window also contains a pull-down menu, the tool bar will be placed directly below the pull-down menu. Multiple tool bars may be added to a window. Although it is not strictly allowed by the MDI specification, a tool bar may also be added to an MDI parent, since many applications do it anyway. The figure below shows a graphical implementation of a UIW_TOOL_BAR class object with various window objects:



The UIW_TOOL_BAR class is declared in UI_WIN.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_TOOL_BAR : public UIW_WINDOW
{
public:
    static ZIL_ICHAR _className[];

    UIW_TOOL_BAR(int left, int top, int width, int height,
        WNF_FLAGS wnFlags = WNF_NO_FLAGS,
        WOF_FLAGS woFlags = WOF_BORDER | WOF_NON_FIELD_REGION,
        WOAF_FLAGS woAdvancedFlags = WOAF_NO_FLAGS);
    virtual ~UIW_TOOL_BAR(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_TOOL_BAR(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
#if defined (ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_TOOL_BAR class, *_className* is "UIW_TOOL_BAR."

UIW_TOOL_BAR::UIW_TOOL_BAR

Syntax

#include <ui_win.hpp>

UIW_TOOL_BAR(int *left,* int *top,* int *width,* int *height,*
WNF_FLAGS *wnFlags* = WNF_NO_FLAGS,
WOF_FLAGS *woFlags* = WOF_BORDER | WOF_NON_FIELD_REGION,
WOAF_FLAGS *woAdvancedFlags* = WOAF_NO_FLAGS);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a new UIW_TOOL_BAR class object.

- $left_{in}$ and top_{in} is the starting position of the tool bar. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the tool bar. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $height_{in}$ is the height of the tool bar. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $wnFlags_{in}$ are flags that define the operation of the tool bar. The following flags (declared in **UI_WIN.HPP)** affect the operation of a UIW_TOOL_BAR class object:

  WNF_BITMAP_CHILDREN—Used to denote that some of the tool bar's sub-objects contain bitmaps.

  WNF_NO_FLAGS—Does not associate any special window flags with the tool bar. This flag should not be used in conjunction with any other WNF flags.

  WNF_NO_WRAP—Causes objects placed in the tool bar to be positioned according to their specified coordinates. By default, objects within a tool bar are automatically positioned so that they are edge-to-edge from left-to-right, in the order in which they were added. If more objects are added than can fit on a single line of the tool bar, the tool bar will wrap and place the remaining objects on the next line. If the **WNF_NO_WRAP** flag is set, however, objects on the tool bar will not be automatically positioned, but will be positioned to the location indicated in their constructor.

  WNF_SELECT_MULTIPLE—Allows more than one object to be selected at a time. This flag must be set if check boxes are added to the tool bar.

- $woFlags_{in}$ are flags (common to all window objects) that determine the general operation of the tool bar. The following flags (declared in **UI_WIN.HPP)** affect the operation of a UIW_TOOL_BAR class object:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the

graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag is set by default in the constructor.

WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is l/10th the size of a normal cell by default.

WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags. This flag is set by default in the constructor.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object. The tool bar will take up the full width of the window but will only be as tall as required to display all the objects attached to the tool bar.

WOF_NON_SELECTABLE—Prevents the object from being selected. If this flag is set, the user will not be able to position on nor select any objects in the tool bar. Typically, the object will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

WOF_SUPPORT_OBJECT—Causes the object to be placed in the parent object's support list. The support list is reserved for objects that are not displayed as part of the user region of the window. If this flag is set on the tool bar it will not be scrolled if the window is scrolled. Also, objects on the window will not be allowed to overwrite the tool bar if they overlap because the window is scrolled or sized smaller. This flag should be set if the tool bar is added to an MDI parent window.

- *woAdvancedFlags*$_{in}$ are flags (common to all window objects) that determine the advanced operation of the tool bar object. The following flags (declared in **UI_-WIN.HPP)** control the advanced operation of a tool bar object:

  WOAF_NO_FLAGS—Does not associate any special advanced flags with the window object. This flag should not be used in conjunction with any other WOAF flags.

WOAF_NON_CURRENT—Prevents the object from becoming current. If this flag is set, users will not be able to select the tool bar from the keyboard. Objects on the tool bar may still be selected using the mouse or a hotkey, but they will not become current.

WOAF_NORMAL_HOT_KEYS—Allows the end-user to select an option using its hotkey by pressing the hotkey by itself, without the <Alt> key otherwise required for selecting with a hotkey.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    ZIL_DATE date;
    ZIL_TIME time;

    // Create a window with a tool bar.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_BORDER
        + new UIW_TITLE(" Sample menus ")
        + &(*new UIW_TOOL_BAR(0,0,20,2)
            + new UIW_BUTTON(0,0,10,"Button")
            + new UIW_STRING(0,0,10,"String")
            + new UIW_DATE (0, 0, 10 , Sedate)
            + new UIW_TIME(0,0,10,&t ime));
    *windowManager + window;



    // The tool bar will automatically be destroyed when the window
    // is destroyed.
}
```

UIW_TOOL_BAR::~UIW_TOOL_BAR

Syntax

#include <ui_win.hpp>

virtual ~UIW_TOOL_BAR(void);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_TOOL_-BAR object. All objects attached to the tool bar will also be destroyed.

UIW_TOOL_BAR::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual function returns the object's class name.

- $returnValue_{out}$ is a pointer to *_className.*

UIW_TOOL_BAR::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the tool bar object. It is declared virtual so that any derived tool bar class can override its default operation.

- *returnValue*$_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the tool bar object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by Event():

  L_DOWN—Moves the focus down one object. If there is no object below the current object, focus will "wrap" to an object at the top of the tool bar and to the right of the current object. This message is interpreted from a keyboard event.

  L_LEFT—Moves the focus left one object. If there is no object to the left of the current object, focus will "wrap" to an object on the right of the tool bar and above the current object. This message is interpreted from a keyboard event.

  L_RIGHT—Moves the focus right one object. If there is no object to the right of the current object, focus will "wrap" to an object on the left of the tool bar and below the current object. This message is interpreted from a keyboard event.

L_UP—Moves the focus up one object. If there is no object above the current object, focus will "wrap" to an object at the bottom of the tool bar and to the left of the current object. This message is interpreted from a keyboard event.

S_ADD_OBJECT—Causes a new object to be added to the tool bar. *event.data* will point to the new object to be added.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_DISPLAY_ACTIVE—Causes the object to draw itself to appear active. An active object is one that is on the active (i.e., current) window. Most objects do not display differently whether they are active or inactive. An active object should not be confused with a current object. An object is active if it is on the active window. However, it may not be the current object on the window.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed by the event overlaps the region of the object. This message is sent by the window to all the non-current, active objects attached to it.

S_DISPLAY_INACTIVE—Causes the object to draw itself to appear inactive. An inactive object is one that is not on the active (i.e., current) window. Most objects do not display differently whether they are inactive or active.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed with the event overlaps the region of the object. This message is sent by the window to all the objects attached to it.

S_MOVE—Causes the object to update its location. The distance to move is contained in the *position* field of UI_EVENT. For example, an *event.position.-*

line of -10 and an *event.position.column* of 15 moves the object 10 lines up and 15 columns to the right.

S_REDISPLAY—Causes the object to redraw.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

All other events are passed by Event() to UIW_WINDOW::Event() for processing.

UIW_TOOL_BAR::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request*, void **data,*
ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- *returnValue*$_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- *request*$_{in}$ is a request to get or set information associated with the object. The following requests (defined in UI_WIN.HPP) are recognized by the window:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself

accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

All other requests are passed by **Information()** to **UIW_WINDOW::Information( )** for processing.

- $data_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- $objectID_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_TOOL_BAR::UIW_TOOL_BAR

Syntax

```
#include <ui_win.hpp>

UIW_TOOL_BAR(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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_TOOL_BAR by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a tool bar object is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objects* is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WlNDOW_OBJECT: .objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TOOL_BAR::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_TOOL_BAR from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TOOL_BAR::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- *name*$_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOWJDBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_TOOL_BAR::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New()** function.

- *returnValue*$_{out}$ is a pointer to the object's **New()** function.

UIW_TOOL_BAR::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- name$_{in}$ the name of the object to be stored.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume* 7.

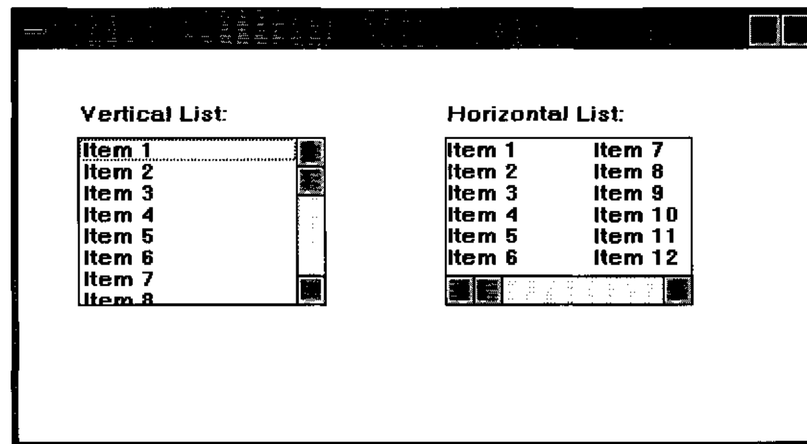
- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see

the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 32 - UIW_VT_LIST

The UIW_VT_LIST class is a selection object used to present a list of objects to the end-user. The objects can be text only or may contain a bitmap or icon. The vertical list will position the objects in a single vertical column. A vertical scroll bar can be added to the list to allow scrolling with the mouse. A typical use for the vertical list is to present a list of items, perhaps file names, and to allow the end-user to select one or more of the items. The figure below shows the graphical implementation of a UIW_VT_LIST object with several string objects:



The UIW_VT_LIST class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_VT_LIST : public UIW_WINDOW
{
public:
    static ZIL_ICHAR _className[];

    UIW_VT_LIST(int left, int top, int width, int height,
        ZIL_COMPARE_FUNCTION compareFunction =
            ZIL_NULLF(ZIL_COMPARE_FUNCTION),
        WNF_FLAGS wnFlags = WNF_NO_WRAP | WNF_CONTINUE_SELECT,
        WOF_FLAGS woFlags = WOF_BORDER,
        WOAF_FLAGS woAdvancedFlags = WOAF_NO_FLAGS);
    UIW_VT_LIST(int left, int top, int width, int height,
        ZIL_COMPARE_FUNCTION compareFunction, WOF_FLAGS flagSetting,
    UI_ITEM   *item)   ;
    virtual ~UIW_VT_LIST(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual void Destroy(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);
#if defined(ZIL_MOTIF)
    virtual void RegionMax(UI_WINDOW_OBJECT *object);
```

```
    virtual ZIL_SCREENID TopWidget(void);
#endif
    virtual void Sort(void);

#if defined (ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void);
    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));
    UIW_VT_LIST(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
#if defined(ZIL_STORE)
    virtual void Store(const ZIL_ICHAR *name, ZIL_STORAGE *file,
        ZIL_STORAGE_OBJECT *object, UI_ITEM *objectTable,
        UI_ITEM *userTable);
#endif

    // List members.
#if defined (ZIL_MACINTOSH)
    UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);
    UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT *object);
    UIW_VT_L1ST &operator+(UI_WINDOW_OBJECT *object);
    UIW_VT_L1ST &operator-(UI_WINDOW_OBJECT *object);
#endif

#if defined (ZIL_MSDOS) || defined (ZIL_CURSES)
public:
    virtual EVENT_TYPE ScrollEvent(UI_EVENT &event);
#endif
};
```

General Members

This section describes those members that are used for general purposes.

_className contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_VT_LIST class, *_className* is "UIW_VT_LIST."

UIW_VT_LIST::UIW_VT_LIST

Syntax

```
#include <ui_win.hpp>

UIW_VT_LIST(int left, int top, int width, int height,
    ZIL_COMPARE_FUNCTION compareFunction =
        ZIL_NULLF(ZIL_COMPARE_FUNCTION),
    WNF.FLAGS wnFlags = WNF_NO_WRAP,
    WOF_FLAGS woFlags = WOF_BORDER,
    WOAF_FLAGS woAdvancedFlags = WOAF_NO_FLAGS);
    or
UIW_VT_LIST(int left, int top, int width, int height,
    ZIL_COMPARE_FUNCTION compareFunction, WOF_FLAGS flagSetting,
    UI_ITEM "item);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These overloaded constructors create a new UIW_VT_LIST class object.

The first overloaded constructor creates a UIW_VT_LIST object.

- $left_{in}$ and top_{in} is the starting position of the vertical list. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the vertical list. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $height_{in}$ is the height of the vertical list. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- *compareFunction*$_{in}$ is a programmer defined function that will be called by the library when sorting the list of objects attached to the vertical list, *compareFunction* is called as each individual object is added and if the list is sorted explicitly by calling the **Sort()** function. The objects can be sorted based on any key unique to the object. Pointers to the objects being compared are passed to the *compareFunction,* so any information required to do the sorting needs to be associated with the object. Because the objects can be of any type, even a derived type, the object pointers will need to be typecast in the *compareFunction.*

  The definition of the *compareFunction* is as follows:

  int FunctionName(void **elementl,* void **element*2);

  returnValue$_{out}$ indicates the relative ordering of the two elements. *returnValue* should be negative if *element 1* should be placed in front of *element2,* 0 if the two elements are sorted the same or positive if *element 1* should come after *element2.*

  elementl$_{in}$ is a pointer to the first element to be compared. This void pointer must be typecast according to the type of object being sorted.

  element2$_{iri}$ is a pointer to the second element to be compared. This void pointer must be typecast according to the type of object being sorted.

- *wnFlags*$_{in}$ are flags that define the operation of the vertical list. The following flags (declared in **UI_WIN.HPP)** affect the operation of a UIW_VT_LIST class object:

  WNF_AUTO_SELECT—Causes each object in the list to be automatically selected when it becomes current. This flag is typically used when radio buttons are added to the vertical list.

  WNF_AUTO_SORT—Causes the vertical list options to be sorted in alphabetical order.

  WNF_BITMAP_CHILDREN—Indicates that some of the objects contain bitmaps. Setting this flag will affect the spacing of objects in the list. Normally, objects are spaced according to a pre-determined cell height value. If this flag is set, however, the objects will be spaced according to the actual height of the objects. This flag should be set when adding check boxes or radio buttons to the vertical list.

WNF_CONTINUE_SELECT—Allows the end-user to drag through the list options with the mouse button pressed. If this flag is not set, the highlight on the list options will not follow the dragging mouse.

WNF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WNF_-FLAGS.

WNF_NO_WRAP—Will not allow arrowing up or down to wrap from the end of the list to the beginning or vice versa.

WNF_SELECT_MULTIPLE—Allows more than one object to be selected at a time.

- $woFlags_{in}$ are flags (common to all window objects) that determine the general operation of the vertical list object. The following flags (declared in **UI_WIN.HPP)** control the general presentation of, and interaction with, a UIW_VT_LIST class object:

    WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles.

    WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is 1/10th the size of a normal cell by default.

    WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags.

    WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

    WOF_NON_SELECTABLE—Prevents the object from being selected. Typically, the object will be drawn in such a manner as to appear non-selectable (e.g., it may appear lighter than a selectable field).

- $woAdvancedFlags_{in}$ are flags (general to all window objects) that determine the advanced operation of the vertical list object.

  WOAF_NO_FLAGS—Does not associate any special advanced flags with the window object. This flag should not be used in conjunction with any other WOAF flags.

  WOAF_NON_CURRENT—Prevents the object from becoming current. If this flag is set, users will not be able to select the vertical list from the keyboard. The vertical list may still be selected using the mouse, but it will not become current.

The second overloaded constructor creates a vertical list using a pre-defined item array. These items are used to create UIW_STRING objects.

- $left_{in}$ and top_{in} is the starting position of the vertical list. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the vertical list. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $height_{in}$ is the height of the vertical list. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $compareFunction_{in}$ is a programmer defined function that will be called by the library when sorting the list of objects attached to the vertical list. For more details, see the description of *compareFunction* with the first constructor.

- $flagSetting_{in}$ is a value that is checked against each UI_ITEM's *value* field. If the item's *value* field is the same as *flagSetting,* that item is marked as selected.

- $item_{in}$ is an array of UI_ITEM structures that are used to construct a set of string items within the vertical list. For more information regarding the use of the UI_ITEM structure, see "Chapter 18—UI_ITEM" in *Programmer's Reference Volume 1.*

Example

```
#include <ui_win.hpp>

ExampleFunction1(UI_WINDOW_MANAGER *windowManager)
{
    // Create the list box field.
    UIW_VT_L1ST *listBox = new UIW_VT_LIST(10, 1, 25, 6)
    *listBox
        + new UIW_STRING(0, 0, 19, "Item 1")
        + new UIW_STRING(0, 0, 19, "Item 2")
        + new UIW_STRING(0, 0, 19, "Item 3")
        + new UIW_STRING(0, 0, 19, "Item 4");

    // Attach the list box to the window.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + UIW_BORDER
        + new UIW_TITLE("Sample list box")
        + new UIW_PROMPT(2, 1, "List box:")
        + listBox;
    *windowManager + window;



    // The list box will automatically be destroyed when the window
    // is destroyed.
}

ExampleFunction2(UI_WINDOW_MANAGER *windowManager)
{
    UI_ITEM listBoxItems[] =
    {
        { 11,   NULL,   "Item 1.1", STF_NO_FLAGS },
        { 12,   NULL,   "Item 1.2", STF_NO_FLAGS },
        { 21,   NULL,   "Item 2.1", STF_NO_FLAGS },
        { 22,   NULL,   "Item 2.2", STF_NO_FLAGS },
        { 0,    NULL,   NULL, 0 }
    };

    // Create the window.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + UIW_BORDER
        + new UIW_TITLE("Sample list box")
        + new UIW_PROMPT(2, 1, "List box:")
        + new UIW_VT_IiIST(10, 1, 20, 6, NULL, WOF_NO_FLAGS, listBoxItems);
    *windowManager + window;



    // The list box will automatically be destroyed when the window
    // is destroyed.
}
```

UIW_VT_LIST::~UIW_VT_LIST

Syntax

```
#include <ui_win.hpp>

virtual ~UIW_VT_LIST(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_VT_LIST object. All objects attached to the vertical list will also be destroyed.

UIW_VT_LIST::Add

Syntax

```
#include <ui_win.hpp>

UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);
    or
UIW_VT_LIST &operator + (UI_WINDOW_OBJECT *object);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function is used to add an object to the vertical list.

- *returnValue*$_{out}$ is a pointer to *object* if the addition was successful. Otherwise, *returnValue* is NULL.

- *object*$_{in}$ is a pointer to the object to be added to the vertical list.

The second overloaded operator adds an item to the UIW_VT_LIST. This operator overload is equivalent to calling the Add() function, except that it allows the chaining of item additions to the UIW_VT_LIST.

- *returnValue*$_{out}$ is a pointer to the UIW_VT_LIST object. This pointer is returned so that the operator may be used in a statement containing other operations.

- *object*$_{in}$ is a pointer to the item that is to be added to the list.

UIW_VT_LIST::ClassName

Syntax

#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to *_className.*

UIW_VT_LIST::Destroy

Syntax

#include <ui_win.hpp>

virtual void Destroy (void);

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function destroys all the objects attached to the vertical list.

UIW_VT_LIST::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function processes run-time messages sent to the vertical list object. It is declared virtual so that any derived vertical list class can override its default operation.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time message for the vertical list object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event():**

  L_BEGIN_SELECT—Indicates that the end-user began the selection of the object by pressing the mouse button down while on the object.

  L_BOTTOM—Scrolls the list to the last page and makes the last item in the list current. This message is interpreted from a keyboard event.

  L_CONTINUE_SELECT—Indicates that the end-user previously clicked down on the object with the mouse and is now continuing to hold the mouse button down while on the object.

  L_DOUBLE_CLICK—Indicates that the end-user double-clicked on an object with the mouse.

  L_DOWN—Moves the focus down one object. If the current object is at the bottom of the list and the WNF_NO_WRAP flag is not set, focus will move to the top item in the list. This message is interpreted from a keyboard event.

  L_END_SELECT—Indicates that the selection process, initiated with the L_-BEGIN_SELECT message, is complete. For example, the end-user has pressed and released the mouse button.

  L_LEFT—Moves the focus up one object. If the current object is at the top of the list and the WNF_NO_WRAP flag is not set, focus will move to the bottom item in the list. This message is interpreted from a keyboard event.

  L_NEXT—The list object processes this message by suppressing it. This allows the list's parent window to process it. This message is interpreted from a keyboard event.

  L_PGDN—Causes the list to scroll down a page. This message is interpreted from a keyboard event.

  L_PGUP—Causes the list to scroll up a page. This message is interpreted from a keyboard event.

L_PREVIOUS—The list object processes this message by suppressing it. This allows the list's parent window to process it. This message is interpreted from a keyboard event.

L_RIGHT—Moves the focus down one object. If the current object is at the bottom of the list and the WNF_NO_WRAP flag is not set, focus will move to the top item in the list. This message is interpreted from a keyboard event.

L_SELECT—Indicates that an object on the list has been selected.

L_TOP—Scrolls the list to the first page and makes the first item in the list current. This message is interpreted from a keyboard event.

L_UP—Moves the focus up one object. If the current object is at the top of the list and the WNF_NO_WRAP flag is not set, focus will move to the bottom item in the list. This message is interpreted from a keyboard event.

L_VIEW—Indicates that the mouse is being moved over the list. This message allows the list to alter the mouse image.

S_ADD_OBJECT—Causes a new object to be added to the list, *event.data* will point to the new object to be added.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_CURRENT—Causes the object to draw itself to appear current. This message is sent by the Window Manager to the window when it becomes current. The window, in turn, passes this message to the object on the window that is current. The vertical list passes the message to its current item.

S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager sends this message to a window when the window is subtracted from the

Window Manager. The window, in turn, relays the message to all objects attached to it. The vertical list sends the message to all its children.

S_DISPLAY_ACTIVE—Causes the object to draw itself to appear active. An active object is one that is on the active (i.e., current) window. Most objects do not display differently whether they are active or inactive. An active object should not be confused with a current object. An object is active if it is on the active window. However, it may not be the current object on the window.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed by the event overlaps the region of the object.

S_DISPLAY_INACTIYE—Causes the object to draw itself to appear inactive. An inactive object is one that is not on the active (i.e., current) window. Most objects do not display differently whether they are inactive or active.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed with the event overlaps the region of the object.

S_DRAG_COPY_OBJECT—Indicates the user is dragging the object to copy it.

S_DRAG_MOVE_OBJECT—Indicates the user is dragging the object to move it.

S_DROP_COPY_OBJECT—Indicates the user dropped an object to copy it to this object.

S_DROP_MOVE_OBJECT—Indicates the user dropped an object to move it to this object.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When the window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_MOVE—Causes the object to update its location. The distance to move is contained in the *position* field of UI_EVENT. For example, an *event.position.-line* of -10 and an *event.position.column* of 15 moves the object 10 lines up and 15 columns to the right.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another object or window.

S_REDISPLAY—Causes the object to redraw.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

S_SUBTRACT_OBJECT—Causes an object to be subtracted from the list. *event.data* will point to the object to be subtracted.

S_VSCROLL—Causes the list to scroll vertically, *event.scroll.delta* indicates how far to scroll.

S_VSCROLL_CHECK—Causes the list to scroll the current item into view if it is not currently visible.

All other events are passed by **Event()** to UIW_WINDOW**::Event()** for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived Event() function.

U1W_VT_LIST::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request*, void *"data,*
ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- $returnValue_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- $request_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the vertical list:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_DESTROY_LIST—Destroys all non-support objects attached to the list. This request simply calls **Destroy().**

  I_GET_BITMAP_ARRAY—Returns a pointer to the bitmap array of the current object if it has a bitmap. If a bitmap does not exist, NULL is returned. If this message is sent, *data* must be a pointer to **ZIL_UINT8.**

  I_GET_TEXT—Returns a pointer to the text associated with the current object. If this request is sent, *data* should be a doubly-indirected pointer to **ZIL_ICHAR.** This request does not copy

  I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

  I_SET_BITMAP_ARRAY—Sets the bitmap array associated with the current object, if it has a bitmap. If this message is sent, *data* must be a pointer to an array of **ZIL_UINT8** that contains the object's new bitmap.

  I_SET_TEXT—Sets the text associated with the current object. This request will also redisplay the object with the new text, *data* should be a pointer to the new text.

  All other requests are passed by **Information()** to **UIW_WINDOW::Information()** for processing.

- *data*$_{in/out}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- *objectID*$_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID*, the object will attempt to interpret the request with the *objectID* of the actual object type.

UIW_VT_LIST::RegionMax

Syntax

```
#include <ui_win.hpp>

virtual void RegionMax(UI_WINDOW_OBJECT *object);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function calculates how much space *object* can occupy within the vertical list and sets *object->trueRegion* accordingly.

- *object*$_{in}$ is a pointer to the object that is requesting the maximum region of the vertical list. Its *trueRegion* member will be modified with its actual position.

UIW_VT_LIST::ScrollEvent

Syntax

#include <ui_win.hpp>

virtual EVENT_TYPE ScrollEvent(UI_EVENT *&event);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function handles events related to scrolling the vertical list. Any events that may result in the list's scroll region getting updated (e.g., S_CREATE, L_SIZE) will call this function to update the scroll information. This function is used by OpenZinc. The programmer typically will not call this function.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time scrolling message for the vertical list object. The type of operation performed depends on the event. The following logical events are processed by ScrollEvent():

  S_SCROLLRANGE—Updates the scroll values maintained in *scroll* and *vScrollInfo.* This event also updates the scroll bar's information, if one exists.

  S_VSCROLL_CHECK—Causes the list to scroll the current item into view if it is not currently visible.

  S_VSCROLL_WINDOW—Causes the objects on the vertical list to scroll vertically, *event.scrolldelta* should contain the amount to scroll.

UIW_VT_LIST::Sort

Syntax

```
#include <ui_gen.hpp>

void Sort(void);
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function sorts the UIW_VT_LIST object using the *compareFunction* that was assigned in the constructor. If the list has no compare function, no sort occurs.

UIW_VT_LIST::Subtract
UIW_VT_LIST::operator -

Syntax

```
#include <ui_gen.hpp>

UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT *object);
    or
UIW_VT_LIST &operator - (UI_WINDOW_OBJECT *object);
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

These functions remove an object from the UIW_VT_LIST.

The first function removes an object from the UIW_VT_LIST but does not call the destructor associated with the object. The programmer is responsible for deletion of each object explicitly subtracted from a list.

- *returnValue*$_{out}$ is a pointer to the next item in the list. This value is NULL if there are no more items after the subtracted item.

- *element*$_{in}$ is a pointer to the item to be subtracted from the list.

The second overloaded operator removes an item from the UIW_VT_LIST but does not call the destructor associated with the object. This operator overload is equivalent to calling the Subtract() function, except that it allows the chaining of item removals from the UIW_VT_LIST.

- *returnValue*$_{out}$ is a pointer to the UIW_VT_LIST object. This pointer is returned so that the operator may be used in a statement containing other operations.

- *object*$_{in}$ is a pointer to the item that is to be subtracted from the list.

UIW_VT_LIST::TopWidget

Syntax

#include <ui_win.hpp>

virtual ZIL_SCREENID TopWidget(void);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns the ZIL_SCREENID of the object's top-most Motif Widget if the object is made up of multiple Widgets.

- $returnValue_{out}$ is the ZIL_SCREENID of the *comboShell* if the object is a combo box. Otherwise this function calls UI_WINDOW_OBJECT::TopWidget() and returns the result.

Storage Members

This section describes those class members that are used for storage purposes.

UIW_VT_LIST::UIW_VT_LIST

Syntax

#include <ui_win.hpp>

UIW_VT_LIST(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));

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_VT_LIST by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a vertical list is stored in a data file it is usually stored as part of a UIW_WINDOW and will be loaded when the window is loaded.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UIJWIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:-.userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_VT_LIST::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_VT_LIST from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objects* is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW VT LIST::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:.userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_VT_LIST::NewFunction

Syntax

```
#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's **New( )** function.

- *returnValue*$_{out}$ is a pointer to the object's **New()** function.

UIW_VT_LIST::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- $name_{in}$ is the name of the object to be stored.

- $file_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOWJOB JECT::objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

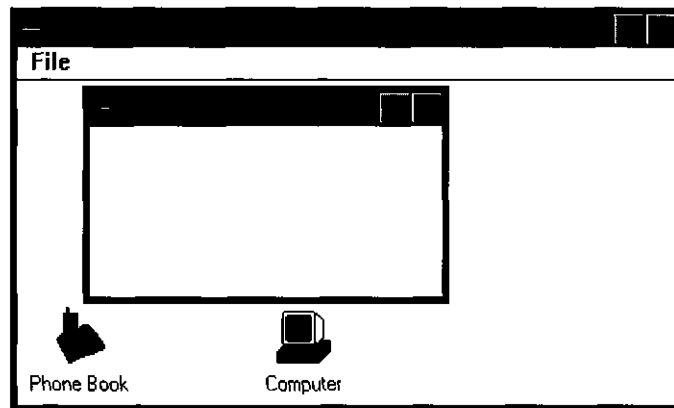
- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the

description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 33 - UIW_WINDOW

The UIW_WINDOW class is used as the controlling object for objects displayed on the screen. Any object derived from UI_WINDOW_OBJECT, including other windows, may be added to a window (e.g., buttons, menus, windows).

The figure below shows a graphical implementation of a UIW_WINDOW class object with an MDI child window and several minimized objects:



A window can be sized, moved, maximized (in those environments that allow it), minimized (in those environments that allow it) and closed. If scrolling is desired, perhaps because more objects are added to the window than can fit in the window, simply adding scroll bars to the window will allow this type of functionality. Geometry management constraints can be placed on objects on the window, if desired.

The end-user can move to a new object on a window by clicking on the object with the mouse, tabbing to the object or arrowing to the object by using the arrow keys on the keyboard. The term tabbing, as used here, generically refers to the process whereby the end-user presses a certain key or key combination and focus moves from object to object in a specific order. Typically, tabbing is accomplished using the <Tab> key, but some environments may require a <Ctrl-Tab> or some other key combination. Arrowing is limited when encountering editable fields. Because an editable field uses arrow keys to maneuver the edit cursor, it may not be possible to move from an editable field using the arrow keys. In these situations either the mouse or tabbing will need to be used.

A window can be added to the Window Manager so that it comes up maximized or minimized initially. To do this, create the window, but before adding it to the Window

Manager, set the WOS_MINIMIZED or WOS_MAXIMIZED status in the window's *woStatus* member. When the window is added to the Window Manager, it will appear as directed by the status flag that was set. NOTE: In Motif, a window cannot be added to the Window Manager in a maximized state. On the Macintosh, a window cannot be added to the Window Manager in a minimized state.

OpenZinc supports MDI windows in all environments. MDI is a TLA (Three-Letter Acronym) for Multiple Document Interface, a concept made popular in the MS-Windows environment. The MDI specification states that MDI child windows added to an MDI parent window clip to the parent window's region. Any number of MDI children can be added to the MDI parent. An MDI parent window must have a pull-down menu (created using the UIW_PULL_DOWN_MENU object). NOTE: OpenZinc has attempted to adhere to the MDI specification as closely as possible. However, MDI applications are not standard across environments. For example, on Motif, Macintosh and NEXTSTEP, MDI children will not be restricted to the region of their parent. They will still be closed if the parent window is closed, but because of operating system architectural and design issues, MDI children will be free to move and size anywhere. Also, in MS-Windows child windows cannot have a pull-down menu, while in all other environments, they can.

If a child window is added to another window, but the windows are not MDI windows, the child should not be movable or sizable. If the child is moved or sized so it overlaps other objects, there may be some ambiguity as to which object draws on top of which and which object gets events. The situation is similar to adding a string so that it overlaps a button. The clipping methods on each environment are different, so the resulting display will be unpredicatable.

The UIW_WINDOW class is declared in UI_WIN.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UIW_WINDOW : public UI_WINDOW_OBJECT, public UI_LIST
{
public:
    static ZIL_ICHAR _className[];
    static int defaultlnitialized;
    WNF_FLAGS wnFlags;
    UI_LIST support;

    UIW_WINDOW(int left, int top, int width, int height,
        WOF_FLAGS woFlags = WOF_NO_FLAGS,
        WOAF_FLAGS woAdvancedFlags = WOAF_NO_FLAGS,
        UI_HELP_CONTEXT helpContext = NO_HELP_CONTEXT,
        UI_WINDOW_OBJECT *minObject = ZIL_NULLP(UI_WINDOW_OBJECT));
    virtual ~UIW_WINDOW(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual void Destroy(void);
    virtual EVENT_TYPE Event (const UI_EVENT Seventh-
    static 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);
    UI_WINDOW_OBJECT *Get(const ZIL_ICHAR *name);
    UI_WINDOW_OBJECT *Get(ZIL_NUMBERID _numberID);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);
    virtual EVENT_TYPE ScrollEvent(UI_EVENT &event);
    static int StringCompare(void *object1, void *object2);

#if defined(ZIL_LOAD)
    virtual ZIL_NEW_FUNCTION NewFunction(void) ;
    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));
    UIW_WINDOW(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));
    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

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

    // List members.
    UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);
    UI_WINDOW_OBJECT *Current(void);
    UI_WINDOW_OBJECT *First(void);
    UI_WINDOW_OBJECT *Last(void);
    UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT *object);
    UIW_WINDOW &operator+(UI_WINDOW_OBJECT *object);
    UIW_WINDOW &operator-(UI_WINDOW_OBJECT *object);

    void SetLanguage(const ZIL_ICHAR *languageName) ;

protected:
    UI_REGION scroll;
    UI_REGION_LIST clipList;
    UI_WINDOW_OBJECT *vScroll;
    UI_WINDOW_OBJECT *hScroll;
    UI_SCROLL_INFORMATION hScrollInfo;
    UI_SCROLL_INFORMATION vScrollInfo;
    ZIL_ICHAR *compareFunctionName;     // Used for storage purposes only.
    UI_WINDOW_OBJECT *defaultobject;
    const ZIL_LANGUAGE *myLanguage;

    void CheckSelection(UI_WINDOW_OBJECT *selectedObject =
        ZIL_NULLP(UI_WINDOW_OBJECT));
    virtual EVENTJYPE Drawltem(const UI_EVENT &event, EVENTJYPE ccode);
    virtual void RegionMax(UI_WINDOW_OBJECT *object);


#ifdef ZIL_MOTIF
    ZIL_UINT32 supportDecorations;
#endif
};
```

General Members

This section describes those members that are used for general purposes.

- *_className* contains a string identifying the class. The string is always the same name as the class, is always in English, and never changes. For example, for the UIW_WINDOW class, *_className* is "UIW_WINDOW."

- *defaultlnitialized* indicates if the default language strings for this object have been set up. The default strings are located in the file **LANG_DEF.CPP.** If *defaultlnitialized* is TRUE, the strings have been set up. Otherwise they have not been. *default-Initialized* is set to TRUE when the strings are set up in the object's constructor.

- *wnFlags* are flags that define the operation of the window. The following flags (declared in **UI_WIN.HPP)** affect the operation of a UIW_WINDOW class object:

  WNF_NO_FLAGS—Does not associate any special window flags with the window object. This flag should not be used in conjunction with any other WNF_FLAGS.

  WNF_NO_SCROLL—Will not allow the window to be scrolled.

  WNF_NO_WRAP—Will not allow tabbing to wrap from the end of a window to the beginning or vice versa.

  WNF_SELECT_MULTIPLE—Allows more than one object to be selected at a time. This flag must be set if check boxes are added directly to the window.

- *support* is a special list of objects that have been added to the window. The support list is reserved for objects that do not appear in the user region of the window. These objects typically include the border, title, system button, maximize button and minimize button. If the window has them, the pull-down menu and status bar will typically be support objects as well. Because these objects are not part of the user region of the window, they will not be scrolled if the window is scrolled. Nor will they be overwritten if the window is sized smaller so that some objects overlap them. An object is placed in the support list if its WOF_SUPPORT_OBJECT flag is set when it is added to the window. Care should be taken when creating a support object. Generally an object should only be created as a support object if the flag is set by default in the constructor. Undesirable effects may result when setting the flag on other objects.

- *scroll* is the entire, virtual scroll region of the window.

- *clipList* is a list of the regions occupied by objects in the window. *clipList.First( )* will return a UI_REGION_ELEMENT that contains the user region of the window. The regions occupied by non-field region objects are maintained in *clipList.*

- *vScroll* is a pointer to the vertical scroll bar attached to the window, if one exists.

- *hScroll* is a pointer to the horizontal scroll bar attached to the window, if one exists.

- *vScrollInfo* is the vertical scroll information for the window. This information is kept so that a window may be scrolled without scroll bars.

- *hScrollInfo* is the horizontal scroll information for the window. This information is kept so that a window may be scrolled without scroll bars.

- *compareFunctionName* is the string representation of the compare function's name. It is used for storage purposes only.

- *defaultObject* is a pointer to the default button on the window, if one exists. The default button is set by creating a button with the BTF_DEFAULT_BUTTON flag set.

- *myLanguage* is the ZIL_LANGUAGE object that contains the string translations for this object.

- *supportDecorations* indicates which X decorations have been associated with the window. X decorations include such things as a system button, maximize button, minimize button, title and border. If *supportDecorations* is 0, no decorations have been associated with the window. This member is specific to Motif.

UIW_WINDOW::UIW_WINDOW

Syntax

#include <ui_win.hpp>

UIW_WINDOW(int *left,* int *top,* int *width,* int *height,*
    WOF_FLAGS *woFlags* = WOF_NO_FLAGS,
    WOAF_FLAGS *woAdvancedFlags* = WOAF_NO_FLAGS,
    UI_HELP_CONTEXT *helpContext* = NO_HELP_CONTEXT,

UI_WINDOW_OBJECT *minObject = ZIL_NULLP(UI_WINDOW_OBJECT);

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This constructor creates a new UIW_WINDOW class object.

- $left_{in}$ and top_{in} is the starting position of the window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the window. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $height_{in}$ is the height of the window. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $woFlags_{in}$ are flags (common to all window objects) that determine the general operation of the window. The following flags (declared in UI_WIN.HPP) affect the operation of a UIW_WINDOW class object:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag should not be used if the window has a UIW_BORDER object.

  WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying

an object's position in minicell coordinates. A minicell is l/10th the size of a normal cell by default.

WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags. This flag is set by default in the constructor.

WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

- *woAdvancedFlags*$_{in}$ are flags (common to all window objects) that determine the advanced operation of the window. The following flags (declared in **UI_WIN.HPP)** control the advanced operation of a window:

  WOAF_DIALOG_OBJECT—Creates the window as a dialog box. A dialog box is a temporary window used to display or receive information from the user. Using this flag will cause a special dialog style border to be displayed.

  NOTE: Some operating environments (e.g, Windows) will create a border, system button and title for a dialog window. Other environments (e.g., DOS) may not, and so a border, system button and title must be added to the dialog window by the programmer. OpenZinc will ignore any support objects in environments that automatically provide them, such as Windows.

  WOAF_LOCKED—Prevents the Window Manager from removing the window from the display. The WOAF_LOCKED flag must be cleared before the Window Manager will allow the window to be removed from the display.

  WOAF_MDI_OBJECT—Causes the window to be an MDI window. If this flag is set on a window that is added to the Window Manager, it becomes an MDI parent (i.e., it can contain MDI child objects). An MDI parent must have a pull-down menu. An MDI parent should contain only support objects (i.e., system button, border, title, etc.), the required pull-down menu, an optional tool bar and MDI children.

  If this flag is set on a window that is added to another MDI window, it becomes an MDI child window. MDI child windows can be moved or sized but will remain entirely within the MDI parent window.

  NOTE: MDI is not standard across environments. For example, in Windows, DOS, Curses and OS/2, child windows will be clipped by their parent window, but in Motif, NEXTSTEP and Macintosh, the child windows will not be clipped

by their parent. In these environments, the child windows are still owned by the parent window, however, so closing the parent window will cause all child windows added to the parent to close also.

WOAF_MODAL—Prevents any other window from receiving events from the Window Manager. A modal window receives all events until it is removed from the display.

WOAF_NO_DESTROY—Prevents the window from being destroyed when it is closed. If this flag is set, the window can be removed from the display, but the programmer is responsible for destroying the window.

WOAF_NO_FLAGS—Does not associate any special advanced flags with the window object. This flag should not be used in conjunction with any other WOAF flags. This flag is set by default in the constructor.

WOAF_NO_MOVE—Prevents the end-user from changing the screen location of the window at run-time. This flag must be set if the window is to be a non-MDI child.

WOAF_NO_SIZE—Prevents the end-user from changing the size of the window at run-time. This flag must be set if the window is to be a non-MDI child.

WOAF_NORMAL_HOT_KEYS—Allows the end-user to select an option using its hotkey by pressing the hotkey by itself, without the <Alt> key otherwise required for selecting with a hotkey. Care should be taken when using this flag on a window, as editable objects will no longer work properly.

WOAF_TEMPORARY—Causes the window to be displayed temporarily. If another window is made current or a non-temporary window is added to the Window Manager, all temporary windows are removed automatically by the Window Manager.

- *helpContext*$_{in}$ identifies the help information associated with the window. *helpContext* is a help context that was created for the help system. If the end-user presses the help key while on this window, and the current object on the window does not have a help context, this help context will be used to display help. For more information about the help system and help context information see "Chapter 17—UI_HELP_-SYSTEM" of *Programmer's Reference Volume 1.*

- *minObject*$_{in}$ is the UIW_ICON to be displayed when the window is minimized.

Example 1

```
#include <ui_win.hpp>

ExampleFunction1(UI_WINDOW_MANAGER *windowManager)
{
    // Create a window with basic window objects.
    UIW_WINDOW *window = new UIW_WINDOW(0, 1, 67, 11);

    *window
        + new UIW_BORDER
        + new UIW_MAXIMIZE_BUTTON
        + new UIW_MINIMIZE_BUTTON
        + new UIW_SYSTEM_BUTTON
        + new UIW_TITLE("Window 1");

    *windowManager + window;



    // The window will automatically be destroyed when the window
    // manager is destroyed.
}
```

Example 2

```
#include <ui_win.hpp>

ExampleFunction2(UI_WINDOW_MANAGER *windowManager)
{
    // Create a window with basic window objects.
    UIW_ICON *icon = new UIW_ICON(C), 0, "iconLogo", "OpenZinc Logo",
        ICF_MINIMIZE_OBJECT);

    UIW_WINDOW *window = new UIW_WINDOW(0, 1, 67, 11, WOF_NO_FLAGS,
        WOAF_NO_FLAGS, NO_HELP_CONTEXT, icon);

    *window
        + new UIW_BORDER
        + new UIW_MAXIMIZE_BUTTON
        + new UIW_MINIMIZE_BUTTON
        + new UIW_SYSTEM_BUTTON
        + new UIW_TITLE("Window 1");

    *windowManager + window;



    // The window will automatically be destroyed when the window
    // manager is destroyed.
}
```

UIW_WINDOW::~UIW_WINDOW

Syntax

```
#include <ui_win.hpp>

virtual ~UIW_WINDOW(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the UIW_WINDOW object. All objects attached to the window will also be destroyed.

UIW_WINDOW::Add
UIW_WINDOW::operator +

Syntax

```
#include <ui_win.hpp>

UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);
    or
UIW_WINDOW &operator + (UI_WINDOW_OBJECT *object);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These overloaded functions are used to add an object to the window. The order in which objects are added to a window is important because it sets the tabbing order for the window.

The first function adds an object to the UIW_WINDOW.

- *returnValue*$_{out}$ is a pointer to *object* if the addition was successful. Otherwise, *returnValue* is NULL.

- *object*$_{in}$ is a pointer to the object to be added to the window.

The second overloaded operator adds an object to the UIW_WINDOW. This operator overload is equivalent to calling the UIW_WINDOW::Add( ) function except that it allows the chaining of object additions to the UIW_WINDOW.

- *returnValue*$_{out}$ is a pointer to the UIW_WINDOW. This pointer is returned so that the operator may be used in a statement containing other operations.

- *object*$_{in}$ is a pointer to the object that is to be added to the window.

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Create a new window and attach it to the window manager.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_BORDER
        + new UIW_MAXIMIZE_BUTTON
        + new UIW_MINIMIZE_BUTTON
        + new UIW_SYSTEM_BUTTON
        + new UIW_TITLE("Window 1");
    *windowManager + window;
```

UIW_WINDOW::CheckSelection

Syntax

```
#include <ui_win.hpp>

void CheckSelection(UI_WINDOW_OBJECT *selectedObject =
    ZIL_NULLP(UI_WINDOW_OBJECT));
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function is used to update the selected status of objects on the window. If a window does not have the WNF_SELECT_MULTIPLE then only one object on the window can be selected at a time. This function makes sure that the selected status of the objects on the window are set appropriately.

- *selectedObject*$_{in}$ is the object that was selected.

UIW WINDOW::ClassName

Syntax

```
#include <ui_win.hpp>

virtual ZIL_ICHAR *ClassName(void);
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This virtual function returns the object's class name.

- *returnValue*$_{out}$ is a pointer to *_className.*

UIW_WINDOW::Current

Syntax

#include <ui_win.hpp>

UI_WINDOW_OBJECT *Current(void);

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This function returns a pointer to the current object, if one exists, in the window.

- *returnValue*$_{out}$ is a pointer to the current object in the window. If there is no current object, *returnValue* is NULL.

UIW_WINDOW::Destroy

Syntax

#include <ui_win.hpp>

virtual void Destroy (void);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function destroys all the objects attached to the window.

UIW_WINDOW::DrawItem

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows BOS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This virtual advanced function is used to draw the object. If the WOS_OWNERDRAW status is set for the object, this function will be called when drawing the window. This allows the programmer to derive a new class from UIW_WINDOW and handle the drawing of the window, if desired.

- $returnValue_{out}$ is a response based on the success of the function call. If the object is drawn the function returns a non-zero value. If the object is not drawn, 0 is returned.

- $event_{in}$ contains the run-time message that caused the object to be redrawn. *event.region* contains the region in need of updating. The following logical events may be sent to the **DrawItem()** function:

S.CURRENT, S_NON_CURRENT, S_DISPLAY_ACTIVE and **S_DISPLAY_INACTIVE**—Messages that cause the object to be redrawn.

WM_DRAWITEM—A message that causes the object to be redrawn. This message is specific to Windows and OS/2.

Expose—A message that causes the object to be redrawn. This message is specific to Motif.

- *ccode*$_{in}$ contains the logical interpretation of *event.*

UIW_WINDOW::Event

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function processes run-time messages sent to the window object. It is declared virtual so that any derived window class can override its default operation.

- *returnValue*$_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- *event*$_{in}$ contains a run-time message for the window object. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event():**

E_MOUSE—Indicates that a general mouse event occurred. Typically, the window will pass mouse events to the object overlapped by the mouse.

L_BEGIN_SELECT—Indicates that the end-user began the selection of the object by pressing the mouse button down while on the object. The window will route this message to the object that the mouse event occurred on.

L_CANCEL—Causes focus to move from the pull-down menu to the current object on the user region of the window.

L_CONTINUE_SELECT—Indicates that the end-user previously clicked down on the object with the mouse and is now continuing to hold the mouse button down while on the object. The window will route this message to the object that the mouse event occurred on.

L_DOWN—Moves the focus down one object. If there is no object below the current object, focus will "wrap" to an object at the top of the window and to the right of the current object. This message is interpreted from a keyboard event.

L_END_SELECT—Indicates that the selection process, initiated with the L_-BEGIN_SELECT message, is complete. For example, the end-user has pressed and released the mouse button. The window will route this message to the object that the mouse event occurred on.

L_FIRST—Causes the first object on the window to be made current.

L_HELP—Causes the help system to be displayed. The window passes this message to the current object to let it display its help. If the current object does not process the message, the window's help context will be displayed.

L_LAST—Causes the last object on the window to be made current.

L_LEFT—Moves the focus left one object. If there is no object to the left of the current object, focus will "wrap" to an object on the right of the window and above the current object. This message is interpreted from a keyboard event.

L_MAXIMIZE—Causes the window to be maximized so that it is as large as allowed. If the window is added to the Window Manager it will be the size of the screen. If the window is an MDI child, it will be as large as its parent's user region. If the window is already in a maximized state, L_MAXIMIZE causes it to return to its original size.

L_MDICHILD_EVENT + L_MOYE—Causes the current MDI child window to go into "move mode." If, for example, the end-user selects the "Move" option from the system menu, the window can then be moved using the arrow keys.

L_MDICHILD_EVENT + L_NEXT_WINDOW—Makes the next MDI child the current MDI child.

L_MDICHILD_EVENT + L_SIZE—Causes the current MDI child window to go into "size mode." If, for example, the end-user selects the "Size" option from the system menu, the window can then be sized using the arrow keys.

L_MINIMIZE—Causes the window to be minimized. If the window has a minimize icon, it will be displayed. If the window is already in a minimized state, L_MINIMIZE causes it to return to its original size.

L_MOVE—Causes the window to go into "move mode." If, for example, the end-user selects the "Move" option from the system menu, the window can then be moved using the arrow keys.

L_NEXT—Causes the next selectable object in the list of window objects to become current. If the last field on the window is current, the first object will become current unless the WNF_NO_WRAP flag is set. This message is interpreted from a keyboard event.

L_PREVIOUS—Causes the previous selectable object in the list of window objects to become current. If the first field on the window is current, the last object will become current unless the WNF_NO_WRAP flag is set. This message is interpreted from a keyboard event.

L_RESTORE—Causes the window to return to its normal size if the window was in a maximized or minimized state.

L_RIGHT—Moves the focus right one object. If there is no object to the right of the current object, focus will "wrap" to an object on the left of the window and below the current object. This message is interpreted from a keyboard event.

L_SIZE—Causes the window to go into "size mode." If, for example, the end-user selects the "Size" option from the system menu, the window can then be sized using the arrow keys.

L_UP—Moves the focus up one object. If there is no object above the current object, focus will "wrap" to an object at the bottom of the window and to the left of the current object. This message is interpreted from a keyboard event.

L_VIEW—Indicates that the mouse is being moved over the window. This message allows the window to alter the mouse image.

S_ADD_OBJECT—Causes a new object to be added to the window, *event.data* will point to the new object to be added.

S_ALT_KEY—Causes focus to move from the user region to the pull-down menu or, if the pull-down menu has focus, from the pull-down menu to the current object on the user region of the window.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_CURRENT—Causes the object to draw itself to appear current. This message is sent by the Window Manager to the window when it becomes current. The window, in turn, passes this message to the object on the window that is current.

S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager sends this message to the window when the window is subtracted from the Window Manager. The window, in turn, relays the message to all objects attached to it.

S_DISPLAY_ACTIVE—Causes the object to draw itself to appear active. An active object is one that is on the active (i.e., current) window. Most objects do not display differently whether they are active or inactive. An active object should not be confused with a current object. An object is active if it is on the active window. However, it may not be the current object on the window.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed by the event overlaps the region of the object. This message is sent by the window to all the non-current, active objects attached to it.

S_DISPLAY_INACTIVE—Causes the object to draw itself to appear inactive. An inactive object is one that is not on the active (i.e., current) window. Most objects do not display differently whether they are inactive or active.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed with the event overlaps the region of the object. This message is sent by the window to all the objects attached to it.

S_DRAG_COPY_OBJECT—Indicates the user is dragging the object to copy it.

S_DRAG_MOVE_OBJECT—Indicates the user is dragging the object to move it.

S_DROP_COPY_OBJECT—Indicates the user dropped an object to copy it to this object.

S_DROP_MOVE_OBJECT—Indicates the user dropped an object to move it to this object.

S_HSCROLL—Causes the window to scroll horizontally, *event.scrolldelta* indicates how far to scroll.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When the window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_MAXIMIZE—Causes the window to be maximized so that it is as large as allowed. If the window is added to the Window Manager it will be the size of the screen. If the window is an MDI child, it will be as large as its parent's user region. If the window is already in a maximized state, S_MAXIMIZE causes it to return to its original size.

S_MDICHILD_EVENT + S_CLOSE—Causes the current MDI child to be closed. The MDI parent will subtract the current MDI child and, if the MDI child does not have the WOAF_NO_DESTROY flag set, will delete the child.

S_MDICHILD_EVENT + S.MAXIMIZE—Causes the current MDI child window to be maximized so that it is as large as its parent's user region. If the window is already in a maximized state, S_MDI_CHILD_EVENT + S_MAXIMIZE causes it to return to its original size.

S_MDICHILD_EVENT + S.MINIMIZE—Causes the current MDI child window to be minimized. If the window has a minimize icon, it will be displayed. If the window is already in a minimized state, S_MDI_CHILD_-EVENT + S_MINIMIZE causes it to return to its original size.

S_MDICHILD_EVENT + S_RESTORE—Causes the current MDI child window to return to its normal size if the window was in a maximized or minimized state.

S_MINIMIZE—Causes the window to be minimized. If the window has a minimize icon, it will be displayed. If the window is already in a minimized state, S_MINIMIZE causes it to return to its original size.

S_MOVE—Causes the object to update its location. The distance to move is contained in the *position* field of UI_EVENT. For example, an *event.position.-line* of -10 and an *event.position.column* of 15 moves the object 10 lines up and 15 columns to the right.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another window.

S_REDISPLAY—Causes the object to redraw.

S_REGION_DEFINE—Causes the object to reserve a region of the screen in which it will display.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

S_RESET_DISPLAY—Changes the display to a different resolution, *event.data* should point to the new display class to be used. If *event.data* is NULL, then a text mode display will be created. This event is specific to DOS and must be

placed on the event queue by the programmer. The library will never generate this event.

S_RESTORE—Causes the window to return to its normal size if the window was in a maximized or minimized state.

S_SCROLLRANGE—Calculates the scroll region for the window.

S_SIZE—Causes the object to recalculate its position and size. When a window is sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_SUBTRACT_OBJECT—Causes an object to be subtracted from the window. *event.data* will point to the object to be subtracted.

S_VSCROLL—Causes the window to scroll vertically, *event.scrolldelta* indicates how far to scroll.

All other events are passed by Event( ) to the current object, if one exists, for processing. If there is no current object, or if the current object cannot process it, the event is passed to UI_WINDOW_OBJECT::Event() for processing.

NOTE: Because most graphical operating systems already process their own events related to this object, the messages listed above may not be handled in every environment. Wherever possible, OpenZinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible. In these situations, the system event can be trapped in a derived Event() function.

UIW_WINDOW::First

Syntax

```
#include <ui_win.hpp>

UI_WINDOW_OBJECT *First(void);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function returns a pointer to the first object, if one exists, in the window.

- $returnValue_{out}$ is a pointer to the first object in the window. If there is no first object, *returnValue* is NULL.

UIW_WINDOW::Generic

Syntax

#include <ui_win.hpp>

static 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);

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh •OSF/Motif • Curses • NEXTSTEP

Remarks

This function creates a new window that includes a border, a maximize button, a minimize button, a generic system button and a title.

- $returnValue_{out}$ is a pointer to the window that was created.

- $left_{in}$ and top_{in} is the starting position of the window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the window. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $height_{in}$ is the height of the window. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $title_{in}$ is the text that is to appear on the window's title bar.

- $minObject_{in}$ is the UIW_ICON to be displayed when the window is minimized.

- $woFlags_{in}$ are flags (common to all window objects) that determine the general operation of the window. The following flags (declared in **UI_WIN.HPP)** affect the operation of a UIW_WINDOW class object:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag should not be used if the window has a UIW_BORDER object.

  WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is l/10th the size of a normal cell by default.

  WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags. This flag is set by default in the constructor.

  WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

- *woAdvancedFlags*$_{in}$ are flags (common to all window objects) that determine the advanced operation of the window. The following flags (declared in **UI_WIN.HPP)** control the advanced operation of a window:

  WOAF_DIALOG_OBJECT—Creates the window as a dialog box. A dialog box is a temporary window used to display or receive information from the user. Using this flag will cause a special dialog style border to be displayed.

  NOTE: Some operating environments (e.g, Windows) will create a border, system button and title for a dialog window. Other environments (e.g., DOS) may not, and so a border, system button and title must be added to the dialog window by the programmer. OpenZinc will ignore any support objects in environments that automatically provide them, such as Windows.

  WOAF_LOCKED—Prevents the Window Manager from removing the window from the display. The WOAFJLOCKED flag must be cleared before the Window Manager will allow the window to be removed from the display.

  WOAF_MDI_OBJECT—Causes the window to be an MDI window. If this flag is set on a window that is added to the Window Manager, it becomes an MDI parent (i.e., it can contain MDI child objects). An MDI parent must have a pull-down menu. An MDI parent should contain only support objects (i.e., system button, border, title, etc.), the required pull-down menu, an optional tool bar and MDI children.

  If this flag is set on a window that is added to another MDI window, it becomes an MDI child window. MDI child windows can be moved or sized but will remain entirely within the MDI parent window.

  NOTE: MDI is not standard across environments. For example, in Windows, DOS, Curses and OS/2, child windows will be clipped by their parent window, but in Motif, NEXTSTEP and Macintosh, the child windows will not be clipped by their parent. In these environments, the child windows are still owned by the parent window, however, so closing the parent window will cause all child windows added to the parent to close also.

  WOAF_MODAL—Prevents any other window from receiving events from the Window Manager. A modal window receives all events until it is removed from the display.

WOAF_NO_DESTROY—Prevents the window from being destroyed when it is close. If this flag is set, the window can be removed from the display, but the programmer is responsible for destroying the window.

WOAF_NO_FLAGS—Does not associate any special advanced flags with the window object. This flag should not be used in conjunction with any other WOAF flags.

WOAF_NO_MOVE—Prevents the end-user from changing the screen location of the window at run-time. This flag must be set if the window is to be a non-MDI child.

WOAF_NO_SIZE—Prevents the end-user from changing the size of the window at run-time. This flag must be set if the window is to be a non-MDI child.

WOAF_NORMAL_HOT_KEYS—Allows the end-user to select an option using its hotkey by pressing the hotkey by itself, without the <Alt> key otherwise required for selecting with a hotkey. Care should be taken when using this flag on a window, as editable objects will no longer work properly.

WOAF_TEMPORARY—Causes the window to be displayed temporarily. **If** another window is made current or a non-temporary window is added to the Window Manager, all temporary windows are removed automatically by the Window Manager.

- *helpContext*$_{in}$ identifies the help information associated with the window. *helpContext* is a help context that was created for the help system. If the end-user presses the help key while on this window, and the current object on the window does not have a help context, this help context will be used to display help. For more information about the help system and help context information see "Chapter 17—UI_HELP_-SYSTEM" of *Programmer's Reference Volume 1.*

Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager)
{
    // Create a window with basic window objects.
    UIW_WINDOW *window = UIW_WINDOW::Generic(0, 1, 67, 11, "Windowl");
    *windowManager + window;
```

```
    // The window will automatically be destroyed when the window
    // manager is destroyed.
}
```

UIW_WINDOW::Get

Syntax

#include <ui_win.hpp>

UI_WINDOW_OBJECT *Get(const ZIL_ICHAR **name);*
    or

UI_WINDOW_OBJECT *Get(ZIL_NUMBERID *_numberID);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These overloaded functions are used to get a pointer to a specific object on the window. They do a depth-first search of the objects attached to the window searching for a match on the identification data specified.

The first overloaded function returns the object whose *stringID* matches *name.*

- $returnValue_{out}$ is a pointer to the object whose *stringID* matches *name.* If no object matches *name,* NULL is returned.

- $name_{in}$ *is* the *stringID* of the object to be located.

The second function returns the object whose *numberID* matches *_numberID.*

- $returnValue_{out}$ is a pointer to the object whose *numberID* matches *_numberID.* If no object matches *_numberID,* NULL is returned.

- $_numberID_{in}$ is the *numberID* of the object to be located.

UIW_WIN DOW::Information

Syntax

#include <ui_win.hpp>

virtual void *Information(ZIL_INFO_REQUEST *request,* void *"data,*
ZIL_OBJECTID *objectID* = ID_DEFAULT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function allows OpenZinc Application Framework objects and programmer functions to get or modify specified information about an object.

- $returnValue_{out}$ is a pointer to the return data that was requested. The type of the return data depends on the request. If the request did not require the return of information, this value is NULL.

- $request_{in}$ is a request to get or set information associated with the object. The following requests (defined in **UI_WIN.HPP)** are recognized by the window:

  I_CHANGED_FLAGS—Informs the object that the programmer has changed some flags associated with the object and that the object should update itself accordingly. This request should be sent after changing an object's flags, particularly if the new flag settings will change the visual appearance of the object.

  I_CHECK_SELECTION—Ensures that the proper selected status is set for objects attached to the window. This request simply causes **CheckSelection()** to be called, *data* must be a pointer to the selected object, *data* is passed to **CheckSelection()** as the *selectedObject* parameter.

  I_CLEAR_FLAGS—Clears the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that

contains the flags to be cleared, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to clear the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_-OBJECT. If the WNF_FLAGS are to be cleared, *objectID* should be ID_-WINDOW. This allows the object to process the request at the proper level. This request only clears those flags that are passed in; it does not simply clear the entire field.

I_COPY_TEXT—Copies the *text* associated with the window's title into a buffer provided by the programmer. If this request is sent, *data* must be the address of a buffer where the title's text will be copied. This buffer must be large enough to contain all of the characters associated with the title and the terminating NULL character.

I_DESTROY_LIST—Destroys all non-support objects attached to the window. This request simply calls **Destroy( ).**

I_GET_CLIPREGION—Returns a pointer to a UI_REGION object that contains the window's user region. If this request is sent, *data* must be a pointer to ULREGION.

I_GET_CURRENT—Returns a pointer to the current object in the window's list. If this request is sent, *data* should be a pointer to UI_WINDOW_OBJECT. If *data* is NULL, a pointer to the current object is returned as *returnValue.*

I_GET_DEFAULT_OBJECT—Returns a pointer to the window's default button. If this request is sent, *data* should be a pointer to UI_WINDOW_-OBJECT. If *data* is NULL, a pointer to the default object is returned as *returnValue.*

I_GET_FIRST—Returns a pointer to the first object in the window's list. If this request is sent, *data* should be a pointer to UI_WINDOW_OBJECT. If *data* is NULL, a pointer to the first object is returned as *returnValue.*

I_GET_FLAGS—Requests the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to obtain the WOF_FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the WNF_FLAGS are desired, *objectID* should be ID_WINDOW. This allows the object to process the request at the proper level.

I_GET_LAST—Returns a pointer to the last object in the window's list. If this request is sent, *data* should be a pointer to UI_WINDOW_OBJECT. If *data* is NULL, a pointer to the last object is returned as *returnValue.*

I_GET_NUMBERID_OBJECT—Returns a pointer to an object whose *numberlD* matches the value in *data*, if one exists. This object does a depth-first search of the objects attached to it, looking for a match of the *numberlD.* If no object has a *numberlD* that matches *data*, NULL is returned. If this message is sent, *data* must be a pointer to a NUMBERID. Programmers should use a window's *numberlD* with caution as it may change at run-time. For more details, see the note accompanying the description of UI_WINDOW_-OBJECT::NumberID() in "Chapter 43—UI_WINDOW_OBJECT" of *Programmer's Reference Volume 1.*

I_GET_STRINGID_OBJECT—Returns a pointer to an object whose *stringID* matches the character string in *data,* if one exists. This object does a depth-first search of the objects attached to it looking for a match of the *stringID.* If no object has a *stringID* that matches *data,* NULL is returned. If this message is sent, *data* must be a pointer to a string.

I_GET_TEXT—Returns a pointer to the text associated with the window's title. If this request is sent, *data* should be a doubly-indirected pointer to ZIL_ICHAR. If *data* is NULL, the title's This request does not copy the text into a new buffer.

I_GET_SUPPORT_CURRENT—Returns a pointer to the current object in the support list.

I_GET_SUPPORT_FIRST—Returns a pointer to the first object in the support list.

I_GET_SUPPORT_LAST—Returns a pointer to the last object in the support list.

I_INITIALIZE_CLASS—Causes the object to initialize any basic information that does not require a knowledge of its parent or sibling objects. This request is sent from the constructor of the object.

I_SET_DEFAULT_OBJECT—Sets the window's default button. If this request is sent, *data* must be a pointer to the new default button.

I_SET_FLAGS—Sets the current flag settings for the object. If this request is sent, *data* should be a pointer to a variable of type UIF_FLAGS that contains the flags to be set, and *objectID* should indicate the type of object with which the flags are associated. For example, if the programmer wishes to set the WOF_-FLAGS of an object, *objectID* should be ID_WINDOW_OBJECT. If the WNF_FLAGS are to be set, *objectID* should be ID_WINDOW. This allows the object to process the request at the proper level. This request only sets those flags that are passed in; it does not clear any flags that are already set.

I_SET_HSCROLL—Sets the horizontal scroll bar pointed to by *hScroll* to the scroll bar pointer passed in *event.data.*

I_SET_TEXT—Sets the text associated with the window's title. This request will also redisplay the object with the new text, *data* should be a pointer to the new text.

I_SET_VSCROLL—Sets the vertical scroll bar pointed to by *vScroll* to the scroll bar pointer passed in *event.data.*

All other requests are passed by **Information**) to **UI_WINDOW_OBJECT::-Information()** for processing.

- *data*$_{injout}$ is used to provide information to the function or to receive the information requested, depending on the type of request. In general, this must be space allocated by the programmer.

- *objectID*$_{in}$ is a ZIL_OBJECTID that specifies which type of object the request is intended for. Because the **Information()** function is virtual, it is possible for an object to be able to handle a request at more than one level of its inheritance hierarchy. *objectID* removes the ambiguity by specifying which level of an object's hierarchy should process the request. If no value is provided for *objectID,* the object will attempt to interpret the request with the *objectID* of the actual object type.

Example

```
#include <ui_win.hpp>
ExampleFunction()
{
    // Update the window's title.
    window-information(I_SET_TEXT, "New Window")
```

UIW_WINDOW::Last

Syntax

#include <ui_win.hpp>

UI_WINDOW_OBJECT *Last(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function returns a pointer to the last object, if one exists, in the window. An MDI parent window reorders the objects in its list dynamically so that the current MDI child window is the last object in the parent's list, the next current MDI child is the next-to-last object in the parent's list, and so forth. So, if the window is an MDI parent, this function will return a pointer to the current MDI child.

- $returnValue_{out}$ is a pointer to the last object in the window. If there is no last object, *returnValue* is NULL.

UIW_WINDOW::RegionMax

Syntax

#include <ui_win.hpp>

virtual void RegionMax(UI_WINDOW_OBJECT *object);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function calculates how much space *object* can occupy within the window and sets *object->trueRegion* accordingly. The regions occupied by objects that have the WOF_-NON_FIELD_REGION flag set, such as the title and system button of a window, are not included in the calculation as their regions are reserved. The regions of any other objects, however, are still available and included in the total region, since these objects can overlap with others.

- $object_{in}$ is a pointer to the object that is requesting the maximum region of the window. Its *trueRegion* member will be modified with its actual position.

UIW_WIN DOW::Scroll Event

Syntax

#include <ui_win.hpp>

virtual EVENT_TYPE ScrollEvent(UI_EVENT *&event);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function handles events related to scrolling the window. Any events that may result in the window's scroll region getting updated (e.g., S_CREATE, L_SIZE) will call this function to update the scroll information. This function is used by OpenZinc. The programmer typically will not call this function.

- $returnValue_{out}$ indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event.* If the event could not be processed, S_UNKNOWN is returned.

- $event_{in}$ contains a run-time scrolling message for the window object. The type of operation performed depends on the event. The following logical events are processed by ScrollEvent( ):

  S_HSCROLL—Causes the window to scroll horizontally, *event.scroll.delta* should contain the amount to scroll. The scroll information is updated and the window's appearance is updated.

  S_HSCROLL_SET—Sets the horizontal scroll information for the window and the attached horizontal scroll bar, if one exists. If this event is sent, *event.scroll* should contain the appropriate values.

  S_HSCROLL_WINDOW—Causes the objects on the window to scroll horizontally, *event.scroll.delta* should contain the amount to scroll.

  S_SCROLLRANGE—Updates the scroll values maintained in *scroll, hScrollInfo* and *vScrollInfo.* This event also updates the scroll bars' information, if they exist.

  S_VSCROLL—Causes the window to scroll vertically, *event.scroll.delta* should contain the amount to scroll. The scroll information is updated and the window's appearance is updated.

  S_VSCROLL_SET—Sets the vertical scroll information for the window and the attached vertical scroll bar, if one exists. If this event is sent, *event.scroll* should contain the appropriate values.

  S_VSCROLL_WINDOW—Causes the objects on the window to scroll vertically, *event.scroll.delta* should contain the amount to scroll.

UIW_WINDOW::SetLanguage

Syntax

```
#include <ui_win.hpp>

void SetLanguage(const ZIL_ICHAR *languageName);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function sets the language to be used by the object. The string translations for the object will be loaded and the object's *myLanguage* member will be updated to point to the new ZIL_LANGUAGE object. By default, the object uses the language identified in the **LANG_DEF.CPP** file, which compiles into the library. (If a different default language is desired, simply copy a **LANG_<ISO>.CPP** file from the OpenZinc\SOURCE\-INTL directory to the \OpenZinc\SOURCE directory, and rename it to **LANG_DEF.CPP** before compiling the library.) The language translations are loaded from the **I18N.DAT** file, so it must be shipped with your application.

- *languageName*$_{in}$ is the two-letter ISO language code identifying which language the object should use.

UIW_WINDOW::StringCompare

Syntax

```
#include <ui_win.hpp>

static int StringCompare(void *object1, void *object2);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This function is used as the compare function if the WNF_AUTO_SORT flag is set on

the window. This function causes the objects to be sorted in ascending alphabetical order when added to the window.

- *returnValue*$_{out}$ indicates the relative positioning of the two objects. *returnValue* is negative if the text associated with *objectl* is alphabetically less than the text associated with *object2*, 0 if the text associated with both objects is the same, or positive if the text associated with *objectl* is alphabetically greater than the text associated with *object2.*

- *objectl*$_{in}$ is a pointer to the first object to be compared.

- *object2*$_{in}$ is a pointer to the second object to be compared.

UIW_WINDOW: Subtract
UIW_WIN DOW::operator -

Syntax

```
#include <ui_win.hpp>

UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT *object);
    or

UIW_WINDOW &operator - (UI_WINDOW_OBJECT *object);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

These overloaded functions are used to subtract an object from the window. These functions do not delete the objects, they merely remove them from the list. The programmer is responsible for destroying any objects explicitly subtracted from the window.

The first function subtracts an object from the UIW_WINDOW.

- *returnValue*$_{out}$ is a pointer to *object* if the subtraction was successful. Otherwise, *returnValue* is NULL.

- *object*$_{in}$ is a pointer to the object to be subtracted from the window.

The <u>second</u> overloaded operator subtracts an object from the UIW_WINDOW. This operator overload is equivalent to calling the UIW_WINDOW::Subtract() function except that it allows the chaining of object subtractions from the UIW_WINDOW.

- *returnValue*$_{out}$ is a pointer to the UIW_WINDOW. This pointer is returned so that the operator may be used in a statement containing other operations.

- *object*$_{in}$ is a pointer to the object that is to be subtracted from the window.

Example 1

```
#include <ui_win.hpp>

ExampleFunction1(UI_WINDOW_OBJECT *object1)
{
    // Construct a window, then add objects to it.
    UIW_WINDOW window1;
    window1.Add(object1);



    // Delete a particular element from a list,
    window1.Subtract(object1);
    delete object1;
```

Example 2

```
ExampleFunction2(UI_WINDOW_OBJECT *object1, UI_WINDOW_OBJECT *object2)
{
    // Construct a window, then add objects to it using the
    // + operator overload.
    UIW_WINDOW *window1 = new UIW_WINDOW(0, 0, 40, 15);
    *window1 + object1 + object2;



    // Move objects from window1 to window2.
    UIW_WINDOW *window2 = new UIW_WINDOW;

    while (window1->First())
    {
        UI_WINDOW_OBJECT *object = window1->First();
        *window1 - object;
```

```
    *window2 + object;
}
```

Storage Members

This section describes those class members that are used for storage purposes.

UIW_WINDOW::UIW_WINDOW

Syntax

#include <ui_win.hpp>

UIW_WINDOW(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));

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced constructor creates a new UIW_WINDOW by loading the object from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1*.

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UIJWIN-DOW_OBJECT" in *Programmer's Reference Volume* |. If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT:. userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_WINDOW::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a UIW_WINDOW from a persistent object data

file. It is called by the persistent constructor and is typically not used by the programmer.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOWJ)BJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_WINDOW::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

UIW_WINDOW::NewFunction

Syntax

#include <ui_win.hpp>

virtual ZIL_NEW_FUNCTION NewFunction(void);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This virtual function returns a pointer to the object's New() function.

- *returnValue*$_{out}$ is a pointer to the object's New() function.

UIW_WINDOW::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- $name_{in}$ is the name of the object to be stored.

- $file_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

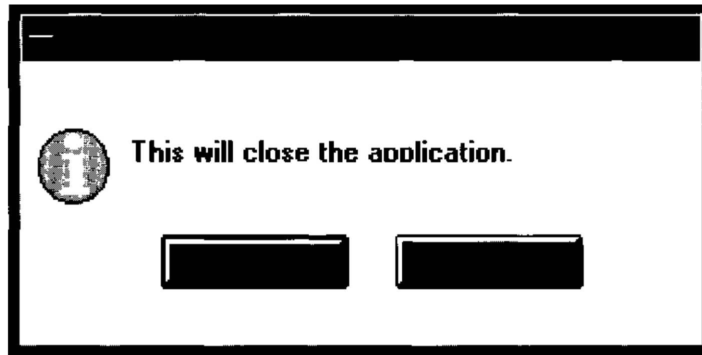
- *object*^ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New() member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 34 - ZAF DIALOG WINDOW

The ZAF_DIALOG_WINDOW class displays a message and one or more response buttons. Program flow halts until the end-user responds to the message by selecting one of the buttons. The buttons are defined by the programmer, but at least one button should place an event on the queue whose value is between DLG_DIALOG_FIRST and DLG_-DIALOG_LAST (e.g., DLG_DIALOG_FIRST + 10). The window will not close and control will not return to the program until such an event is processed. The space from DLG_DIALOG_USER to DLG_DIALOG_LAST is reserved for user defined types. The figure below shows a graphical representation of a typical dialog window:



The ZAF_DIALOG_WINDOW class is declared in UI_WIN.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS ZAF_DIALOG_WINDOW : public UIW_WINDOW
{
public:
    ZAF_DIALOG_WINDOW(int left, int top, int width, int height,
        WOF_FLAGS woFlags = WOF_NO_FLAGS,
        WOAF_FLAGS woAdvancedFlags = WOAF_NO_FLAGS,
        UI_HELP_CONTEXT helpContext = NO_HELP_CONTEXT) ;
    virtual ~ZAF_DIALOG_WINDOW(void);
    EVENT_TYPE Control(void);

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

General Members

This section describes those members that are used for general purposes.

ZAF_DIALOG WINDOW::ZAF_DIALOG_WINDOW

Syntax

#include <ui_win.hpp>

ZAF_DIALOG_WINDOW(int *left,* int *top,* int *width,* int *height,*
WOF_FLAGS *woFlags* = WOF_NO_FLAGS,
WOAF_FLAGS *woAdvancedFlags* = WOAF_NO_FLAGS,
UI_HELP_CONTEXT *helpContext* = NO_HELP_CONTEXT);

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a dialog window object. After the dialog window is created, the window's Control() function should be called. This will display the window and halt program flow until the user responds by selecting a button. The return value from Control() will indicate the user response. The message displayed on the window should be phrased so that the action initiated by selecting one of the response buttons provided will be clear.

- $left_{in}$ and top_{in} is the starting position of the window. Typically, these values are in cell coordinates. If the WOF_MINICELL flag is set, however, these values will be interpreted as minicell values.

- $width_{in}$ is the width of the window. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $height_{in}$ is the height of the window. Typically, this value is in cell coordinates. If the WOF_MINICELL flag is set, however, this value will be interpreted as a minicell value.

- $woFlags_{in}$ are flags (common to all window objects) that determine the general operation of the dialog window. The following flags (declared in **UI_WIN.HPP)** affect the operation of a ZAF_DIALOG_WINDOW class object:

  WOF_BORDER—Draws a border around the object. The graphical border's appearance will depend on the operating system used, and, if in DOS, on the graphics style being used. In text mode, a border may or may not be drawn, depending on the text style being used. See "Appendix A—Support Definitions" in this manual for information on changing DOS graphics mode styles and text mode styles. This flag should not be used if the window has a UIW_BORDER object.

  WOF_MINICELL—Causes the position and size values that were passed into the constructor to be interpreted as minicells. A minicell is a fraction the size of a normal cell. Greater precision in object placement is achieved by specifying an object's position in minicell coordinates. A minicell is 1/10th the size of a normal cell by default.

  WOF_NO_FLAGS—Does not associate any special window flags with the object. This flag should not be used in conjunction with any other WOF flags. This flag is set by default in the constructor.

  WOF_NON_FIELD_REGION—Causes the object to ignore its position and size parameters and use the remaining available space in its parent object.

- $woAdvancedFlags_{in}$ are flags (common to all window objects) that determine the advanced operation of the window. The following flags (declared in **UI_WIN.HPP)** control the advanced operation of a window:

WOAF_DIALOG_OBJECT—Creates the window as a dialog box. A dialog box is a temporary window used to display or receive information from the user. Using this flag will cause a special dialog style border to be displayed.

NOTE: Some operating environments (e.g, Windows) will create a border, system button and title for a dialog window. Other environments (e.g., DOS) may not, and so a border, system button and title must be added to the dialog window by the programmer. OpenZinc will ignore any support objects in environments that automatically provide them, such as Windows.

WOAF_LOCKED—Prevents the Window Manager from removing the window from the display. The WOAF_LOCKED flag must be cleared before the Window Manager will allow the window to be removed from the display.

WOAF_MODAL—Prevents any other window from receiving events from the Window Manager. A modal window receives all events until it is removed from the display.

WOAF_NO_DESTROY—Prevents the window from being destroyed when it is closed. If this flag is set, the window can be removed from the display, but the programmer is responsible for destroying the window.

WOAF_NO_FLAGS—Does not associate any special advanced flags with the window object. This flag should not be used in conjunction with any other WOAF flags. This flag is set by default in the constructor.

WOAF_NO_MOVE—Prevents the end-user from changing the screen location of the window at run-time. This flag must be set if the window is to be a non-MDI child.

WOAF_NO_SIZE—Prevents the end-user from changing the size of the window at run-time. This flag must be set if the window is to be a non-MDI child.

WOAF_NORMAL_HOT_KEYS—Allows the end-user to select an option using its hotkey by pressing the hotkey by itself, without the <Alt> key otherwise required for selecting with a hotkey. Care should be taken when using this flag on a window, as editable objects will no longer work properly.

WOAF_TEMPORARY—Causes the window to be displayed temporarily. **If** another window is made current or a non-temporary window is added to the Window Manager, all temporary windows are removed automatically by the Window Manager.

- *helpContext*$_{in}$ identifies the help information associated with the window. *helpContext* is a help context that was created for the help system. If the end-user presses the help key while on this window, and the current object on the window does not have a help context, this help context will be used to display help. For more information about the help system and help context information see "Chapter 17—UI_HELP_-SYSTEM" of *Programmer's Reference Volume 1.*

ZAF_DIALOG_WINDOW::~ZAF_DIALOG_WINDOW

Syntax

```
#include <ui_win.hpp>

virtual ~ZAF_DIALOG_WINDOW(void);
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This virtual destructor destroys the class information associated with the ZAF_DIALOG_-WINDOW object. All objects attached to the dialog window will also be destroyed.

ZAF_DIALOG_WINDOW::Control

Syntax

```
#include <ui_win.hpp>

EVENT_TYPE Control(void);
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function is used as a control loop for the dialog window. The function adds the window to the Window Manager and does not release control until the window has been closed or an option on the window has been chosen. At least one button should be placed on the window with the BTF_SEND_MESSAGE flag set and a value in the range from DLG_DIALOG_FIRST to DLG_DIALOG_LAST. The window is removed from the Window Manager before the function returns, but is not deleted.

- $returnValue_{out}$ indicates the value of the option selected by the end-user. The value will be in the range from DLG_DIALOG_FIRST to DLG_DIALOG_LAST.

Storage Members

This section describes those class members that are used for storage purposes.

ZAF_DIALOG_WINDOW::ZAF_DIALOG_WINDOW

Syntax

```
#include <ui_win.hpp>

ZAF_DIALOG_WINDOW(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));
```

Portability

This function is available on the following environments:

• DOS Text	• DOS Graphics	• Windows	• OS/2
• Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This advanced constructor creates a new ZAF_DIALOG_WINDOW by loading the object from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1*.

- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1*.

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static **New()** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1*. If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1*. If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

ZAF_DIALOG_WINDOW::Load

Syntax

```
#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a ZAF_DIALOG_WINDOW from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name*$_{in}$ is the name of the object to be loaded.

- *file*$_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume 1.*

- objectTable$_{in}$ is a pointer to a table that contains the addresses of the static **New( )** member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT::objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

ZAF_DIALOG_WINDOW::New

Syntax

```
#include <ui_win.hpp>

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));
```

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This advanced function is used to load a persistent object from a data file. This function is a static class member so that its address can be placed in a table used by the library to load persistent objects from a data file.

NOTE: The application must first create a display if objects are to be loaded from a data file.

- $name_{in}$ is the name of the object to be loaded.

- $file_{in}$ is a pointer to the ZIL_STORAGE_READ_ONLY object that contains the persistent object. For more information on persistent object files, see "Chapter 70—ZIL_STORAGE_READ_ONLY" of *Programmer's Reference Volume 1.*

- *object*$_{in}$ is a pointer to the ZIL_STORAGE_OBJECT_READ_ONLY where the persistent object information will be loaded. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 69—ZIL_STORAGE_OBJECT_READ_ONLY" of *Programmer's Reference Volume L*

- *objectTable*$_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WINDOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- *userTable*$_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

ZAF DIALOG_WINDOW::Store

Syntax

#include <ui_win.hpp>

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

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This advanced function is used to write an object to a data file.

- $name_{in}$ is the name of the object to be stored.

- $file_{in}$ is a pointer to the ZIL_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL_STORAGE" of *Programmer's Reference Volume 1.*

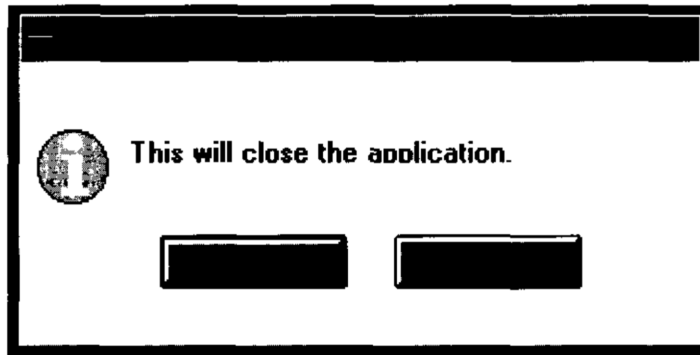
- $object_{in}$ is a pointer to the ZIL_STORAGE_OBJECT where the persistent object information will be stored. This must be allocated by the programmer. For more information on loading persistent objects, see "Chapter 68—ZIL_STORAGE_-OBJECT" of *Programmer's Reference Volume 1.*

- $objectTable_{in}$ is a pointer to a table that contains the addresses of the static New( ) member functions for all persistent objects. For more details about *objectTable* see the description of *UI_WINDOW_OBJECT:.objectTable* in "Chapter 43—UI_WIN-DOW_OBJECT" in *Programmer's Reference Volume 1.* If *objectTable* is NULL, the library will use the object table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

- $userTable_{in}$ is a pointer to a table that contains the addresses of user objects, user functions and compare functions. For more details about *userTable* see the description of *UI_WINDOW_OBJECT::userTable* in "Chapter 43—UI_WINDOW_-OBJECT" in *Programmer's Reference Volume 1.* If *userTable* is NULL, the library will use the user table created by the Designer, if one was linked into the program, or, if no Designer-created table exists, it will use a default empty table.

CHAPTER 35 - ZAF_MESSAGE_WINDOW

The ZAF_MESSAGE_WINDOW class is a type of dialog window that displays a message and one or more response buttons. Program flow halts until the end-user responds to the message by selecting one of the buttons. The response buttons are all pre-defined for use by the message window. The figure below shows a graphical representation of a typical message window:



The ZAF_MESSAGE_WINDOW class is declared in **UI_WIN.HPP.** Its public and protected members are:

```
class ZIL_EXPORT_CLASS ZAF_MESSAGE_WINDOW : public UIW_WINDOW
{
public:
    ZAF_MESSAGE_WINDOW(ZIL_ICHAR *title, ZIL_ICHAR *icon,
        MSG_FLAGS msgFlags, MSG_FLAGS defFlag, ZIL_ICHAR *format, ...);
    virtual ~ZAF_MESSAGE_WINDOW(void);
    EVENT_TYPE Control(void);
};
```

General Members

This section describes those members that are used for general purposes.

ZAF_MESSAGE_WINDOW::ZAF_MESSAGE_WINDOW

Syntax

#include <ui_win.hpp>

ZAF_MESSAGE_WINDOW(ZIL_ICHAR **title*, ZIL_ICHAR **icon,*
MSG_FLAGS *msgFlags,* MSG_FLAGS *defFlag,* ZIL_ICHAR **format, ...);*

Portability

This function is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This constructor creates a message window object. After the message window is created, the window's Control() function should be called. This will display the message window and halt program flow until the user responds by selecting a button. The return value from Control() will indicate the user response. The message displayed on the window should be phrased so that the action initiated by selecting one of the response buttons provided will be clear.

- *title*$_{in}$ is the string to be displayed in the message window's title bar.

- *icon*$_{in}$ is the name of an icon as it is stored in the *UI_WINDOW_OBJECT:.default-Storage* .DAT file or resource file. The icon will be displayed on the window next to the text.

- *rnsgFlags*$_{in}$ specifies which response buttons should be placed on the window. The following flags are available:

  ZIL_MSG_ABORT—Causes a button with the text "Abort" (or the appropriate translation) to be placed on the window.

  ZIL_MSG_CANCEL—Causes a button with the text "Cancel" (or the appropriate translation) to be placed on the window.

ZIL_MSG_HELP—Causes a button with the text "Help" (or the appropriate translation) to be placed on the window.

ZIL_MSG_IGNORE—Causes a button with the text "Ignore" (or the appropriate translation) to be placed on the window.

ZIL_MSG_NO—Causes a button with the text "No" (or the appropriate translation) to be placed on the window.

ZIL_MSG_OK—Causes a button with the text "Ok" (or the appropriate translation) to be placed on the window.

ZIL_MSG_RETRY—Causes a button with the text "Retry" (or the appropriate translation) to be placed on the window.

ZIL_MSG_YES—Causes a button with the text "Yes" (or the appropriate translation) to be placed on the window.

- $defFlag_{in}$ specifies which of the buttons should be the default button. *defFlag* must be one of the flags set in *msgFlags.*

- $format_{in}$ is the **printf** style format string that specifies how the message string is to be displayed.

- $..._{in}$ is the variable-length argument list that contains any arguments required by *format.*

ZAF_MESSAGE_WINDOW::Control

Syntax

```
#include <ui_win.hpp>

EVENT_TYPE Control(void);
```

Portability

This function is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This function is used as a control loop for the message window. The function adds the window to the Window Manager and does not release control until the window has been closed or an option on the window has been chosen. The window is removed from the Window Manager before the function returns, but is not deleted.

- *returnValue*$_{out}$ indicates the value of the option selected by the end-user. The following values can be returned:

  ZIL_DLG_ABORT—Indicates that the Abort button was selected.

  ZIL_DLG_CANCEL—Indicates that the Cancel button was selected.

  ZIL_DLG_HELP—Indicates that the Help button was selected.

  ZIL_DLG_IGNORE—Indicates that the Ignore button was selected.

  ZIL_DLG_NO—Indicates that the No button was selected.

  ZIL_DLG_OK—Indicates that the Ok button was selected.

  ZIL_DLG_RETRY—Indicates that the Retry button was selected.

  ZIL_DLG_YES—Indicates that the Yes button was selected.

APPENDIX A - SUPPORT DEFINITIONS

This appendix describes various support items of OpenZinc Application Framework. The first section lists typedefs and preprocessor variables. The second section lists macros.

Typedefs and Preprocessor Variables

FALSE

This is a boolean value. OpenZinc defines **FALSE** to have a value of zero if it is not already defined by the compiler.

TRUE

This is a boolean value. OpenZinc defines **TRUE** to have a value of one if it is not already defined by the compiler.

ZIL_BAK

This is the default string used as the extension for a backup file.

ZIL_BIGENDIAN

This precompiler variable is defined by the library if the environment's values are interpreted in big-endian fashion. This precompiler variable may need to be defined if porting the library to an unsupported environment.

ZIL_BITMAP_HANDLE

This type is defined to be the bitmap handle type supported by the graphical environment, if any. For example, Windows has an **HBITMAP** type that it uses to process bitmaps. If the graphical environment uses its own bitmap format, it is usually more efficient to use that format and **ZIL_BITMAP_HANDLE** will be defined as the appropriate type. If the graphical environment does not support its own bitmap type, **ZIL_BITMAP_-HANDLE** is simply a pointer to **ZIL_UINT8.**

ZIL_COLOR

This type is defined to be the color type supported by the graphical environment, if any. Some environments might support 24-bit color while others might only support 8-bits.

ZIL_COMPARE_FUNCTION

This typedef is a function with the following signature:

```
typedef int (*ZIL_COMPARE_FUNCTION)(void *, void *);
```

This type of function is used with lists or list objects when user-defined sorting is to be imposed on objects added to the list.

ZIL_DECOMPOSE

Enables character decomposition in Unicode mode. If a Unicode character is composed of a character and one or more modifier characters, in some instances it may need to be split up, or decomposed, into the individual characters for purposes of collating. If this precompiler variable is defined when the library is compiled, the decomposition functionality will be enabled. Otherwise, the library will not allow nor use decomposition. If desired, this precompiler variable should be defined in **UI_ENV.HPP** when compiling the library.

ZIL_D0_FILE_I18N

Allows loading of map tables, language information and locale information from a **.DAT** file. If this precompiler variable is defined, this information will be loaded at run-time. Otherwise, this information is obtained from the default information that was compiled in the library. If desired, this precompiler variable should be defined in **UI_ENV.HPP** when compiling the library. This precompiler variable is defined by default.

ZIL_DO_OS_I18N

Allows loading of language and locale information from the operating system. If this precompiler variable is defined, this information will be obtained from the operating system at run-time. Otherwise, this information is obtained from the default information that was compiled in the library. If desired, this precompiler variable should be defined in **UI_ENV.HPP** when compiling the library. This precompiler variable is defined by default.

ZIL_EXIT_FUNCTION

This typedef is a function with the following signature:

```
typedef EVENT_TYPE (*ZIL_EXIT_FUNCTION)(UI_DISPLAY *,
    UI_EVENT_MANAGER *, UI_WINDOW_MANAGER *);
```

This type of function is used to perform an action when the user attempts to exit the application.

ZIL_EXT

This is the default string used as the extension for a **.DAT** file.

ZIL_HARDWARE

Enables support for non-AT DOS machines, such as the NEC PC 9800 series. If this precompiler variable is defined when the library is compiled, the library will be built to work with non-AT DOS machines. Otherwise, the library will only work with AT compatible machines. If desired, this precompiler variable should be defined in **UI_ENV.HPP** when compiling the library. If this precompiler variable is defined, several source modules will also need to be compiled and linked into the library. In the makefiles in the OpenZinc\SOURCE directory there are definitions for **DOSHARDWAREDEP** and **DOSHARDWARELIB.** By default these variables compile and link in modules for the AT machines (e.g., **i_btcat.obj).** There are similar modules for the NEC machine (e.g., **ijbtcnec.obj)** that should either replace the AT modules if only NEC support is required, or should be used in addition to the AT modules if the executable should support both hardware environments.

ZIL_HOTMARK

This is the character used to identify a hotkey character for an object. By default, **ZIL_HOTMARK** is the '&' character. Thus, if a string passed to an object that supports hotkeys contains an '&' the character immediately following the '&' will be designated as the hotkey for the object. Extreme care should be taken if changing the hotkey designator, as objects stored in **.DAT** files that used a different hotkey designator will no longer work.

ZIL_IBIGNUM

This is a 32-bit signed integer. This type is used with the ZIL_BIGNUM class, which, in turn, is used with the UIW_BIGNUM object.

ZIL_ICHAR

This is a character type that resolves to different types depending on the environment and whether Unicode is supported. In environments that support the Unicode character set, this type may be the wide character type (e.g., **wchar_t)** defined in that environment if the wide character type is known to be properly defined—not all environments that define **wchar_t** define it correctly. If the environment does not support Unicode directly, but OpenZinc is being used in Unicode mode, then this type is **ZIL_UINT16,** a 16-bit unsigned integer type. Otherwise this type resolves to a **char** type. This type should be used instead of **char** or **wchar_t** so that the application will be as portable as possible between compilers and operating systems.

ZIL_ICON_HANDLE

This type is defined to be the icon handle type supported by the graphical environment, if any. For example, Windows has an **HICON** type that it uses to process icons. If the graphical environment uses its own icon format, it is usually more efficient to use that format and **ZIL_ICON_HANDLE** will be defined as the appropriate type. If the graphical environment does not support its own icon type, **ZIL_BITMAP_HANDLE** is simply a pointer to **ZIL_UINT8.**

ZIL_INT8

This is an 8-bit signed type. This type should be used wherever the type must be guaranteed to be 8-bits signed; for example, when storing a value in one environment and reading it in another.

ZIL_INT16

This is a 16-bit signed type. This type should be used wherever the type must be guaranteed to be 16-bits signed; for example, when storing a value in one environment and reading it in another.

ZIL_INT32

This is a 32-bit signed type. This type should be used wherever the type must be guaranteed to be 32-bits signed; for example, when storing a value in one environment and reading it in another.

ZIL_LITTLEENDIAN

This precompiler variable is defined by the library if the environment's values are interpreted in little-endian fashion. This precompiler variable may need to be defined if porting the library to an unsupported environment.

ZIL_LOAD

Enables the load functionality of the storage classes. If this precompiler variable is defined when the library is compiled, the load functionality will be enabled. Otherwise, the load functionality will not be available. If desired, this precompiler variable should be defined in **UI_ENV**.HPP when compiling the library. This precompiler variable is defined by default.

ZIL_MACINTOSH

This precompiler variable is defined by the library when compiling the library or an application for the Macintosh. This precompiler variable can be used to "if def" platform-specific code, if desired.

ZIL_MAXPATHLEN

This is the maximum path length allowed by the operating system for which the library is compiled.

ZIL_MOTIF

This precompiler variable is defined by the library when compiling the library or an application for Motif. This precompiler variable can be used to "if def" platform-specific code, if desired.

ZIL_MOTIF_STYLE

If this precompiler variable is defined, the DOS graphics mode appearance will be similar to the default Motif style. If desired, this precompiler variable should be defined in **UI_ENV.HPP** when compiling the library.

ZIL_MSDOS

This precompiler variable is defined by the library when compiling the library or an application for MS-DOS. This precompiler variable can be used to "if def" platform-specific code, if desired.

ZIL_MSWINDOWS

This precompiler variable is defined by the library when compiling the library or an application for MS-Windows 3.x. This precompiler variable can be used to "if def" platform-specific code, if desired.

ZIL_MSWINDOWS_STYLE

If this precompiler variable is defined, the DOS graphics mode appearance will be similar to the default Windows 3.X style. If desired, this precompiler variable should be defined in **UI_ENV.HPP** when compiling the library. This precompiler variable is defined by default.

ZIL_NEW_FUNCTION

This typedef is a function with the following signature:

```
typedef UI_WINDOW_OBJECT *(*ZIL_NEW_FUNCT10N)(const ZIL_ICHAR *,
    ZIL_STORAGE_READ_ONLY *, ZIL_STORAGE_OBJECT_READ_ONLY *,
    UI_ITEM *, UI_ITEM *);
```

This type of function is used to load persistent objects from a persistent object file.

ZIL_NUMBERID

This typedef is used to give objects a unique value that can be used to identify the object in the context of its parent.

ZIL_OLD_DEFS

Turns on backwards compatibility of names. If an application was written using a previous version of the library, it should still compile if **ZIL_OLD_DEFS** is defined in **UI_ENV.HPP.** It is preferable to upgrade applications rather than use this precompiler variable so that the application can take advantage of current functionality as well as to ensure that future upgrades will be as effortless as possible.

ZIL_OS2

This precompiler variable is defined by the library when compiling the library or an application for OS/2. This precompiler variable can be used to "if def" platform-specific code, if desired.

ZIL_OS2_STYLE

If this precompiler variable is defined, the DOS graphics mode appearance will be similar to the default OS/2 style. If desired, this precompiler variable should be defined in **UI_ENV.HPP** when compiling the library.

ZIL_PATHSEP

This is the character used by the operating system to separate individual path nodes in a path string.

ZIL_POSIX

This precompiler variable is defined by the library when compiling the library or an application for a Posix-compliant environment. This precompiler variable can be used to "if def" platform-specific code, if desired.

ZIL_RBIGNUM

This is a double-precision floating point type. This type is used with the ZIL_BIGNUM class, which, in turn, is used with the UIW_BIGNUM object.

ZIL_SCREENID

This type is used to identify an object or its type as required by the environment. In DOS

and Curses, **ZIL_SCREENID** is used to identify a window region reserved for use by the object. In Windows, Windows NT, OS/2 and Macintosh, **ZIL_SCREENID** is the handle type (e.g., **HWND)** used by the operating system to identify objects. In Motif, **ZIL_SCREENID** is the **Widget** type, used to identify what type of Widget the object is.

ZIL_SHADOW_BORDER

If this precompiler variable is defined, the border on DOS text mode windows will appear as a shadow border. If desired, this precompiler variable should be defined in **UI_ENV.HPP** when compiling the library. This precompiler variable is defined by default.

ZIL_STANDARD_BORDER

If this precompiler variable is defined, the borders on DOS text mode windows will appear as standard single- and double-line borders. If desired, this precompiler variable should be defined in **UI_ENV.HPP** when compiling the library.

ZIL_STORE

Enables the store functionality of the storage classes. If this precompiler variable is defined when the library is compiled, the store functionality will be enabled. Otherwise, the store functionality will not be available. If desired, this precompiler variable should be defined in **UI_ENV.HPP** when building the library. This precompiler variable is defined by default.

ZIL_TEXT_ONLY

Prevents any code that is specific to graphics mode from being compiled into the library. If this precompiler variable is defined when the library is compiled, only text mode code will be put in the library. Otherwise, both text mode and graphics mode code will be placed in the library. If desired, this precompiler variable should be defined in **UI_ENV.HPP** when building the library.

ZIL_3D_BORDER

If this precompiler variable is defined, objects in DOS text mode will have a three-dimensional appearance. However, the objects will require much more screen space. If

desired, this precompiler variable should be defined in **UI_ENV.HPP** when compiling the library.

ZIL_3x_COMPAT

If this precompiler variable is defined, objects in 3.x **.DAT** files will be read in 3.x format. Specifically, date ranges in 4.x are specified differently than in 3.x versions. If this precompiler variable is defined, date ranges will be read assuming the 3.x format and a U.S. format. Otherwise, they will be read assuming they are in the 4.x format, which requires a full year, month and day of month, separated by dashes (i.e.,'-'). If desired, this precompiler variable should be defined in **UI_ENV.HPP** when compiling the library. Old **.DAT** files should be updated as soon as possible to take advantage of 4.x features and to make future upgrades as effortless as possible.

ZIL_UINT8

This is an 8-bit unsigned type. This type should be used wherever the type must be guaranteed to be 8-bits unsigned; for example, when storing a value in one environment and reading it in another.

ZIL_UINT16

This is a 16-bit unsigned type. This type should be used wherever the type must be guaranteed to be 16-bits unsigned; for example, when storing a value in one environment and reading it in another.

ZIL_UINT32

This is a 32-bit unsigned type. This type should be used wherever the type must be guaranteed to be 32-bits unsigned; for example, when storing a value in one environment and reading it in another.

ZIL_UNICODE

Enables Unicode functionality. If this precompiler variable is defined when the library is compiled, the Unicode functionality will be enabled. Otherwise, the library will be compiled for 8-bit character use. If desired, this precompiler variable should be defined in **UI_ENV.HPP** when compiling the library.

ZIL_USER_FUNCTION

This typedef is a function with the following signature:

```
typedef EVENT_TYPE (*ZIL_USER_FUNCTION)(UI_WINDOW_OBJECT *,
    UI_EVENT &, EVENT_TYPE);
```

This type of function is used to perform an action when an object with which the function is associated is acted on.

ZIL_WINNT

This precompiler variable is defined by the library when compiling the library or an application for MS-Windows NT. This precompiler variable can be used to "if def" platform-specific code, if desired.

Macros

AbsValue

Syntax

```
include <ui_env.hpp>

#define AbsValue(arg) ((arg) > 0 ? (arg) : -(arg))
```

Portability

This macro is available on the following environments:

DOS Text • DOS Graphics • Windows			• OS/2
Macintosh	• OSF/Motif	• Curses	• NEXTSTEP

Remarks

This macro returns the absolute value of *arg.*

Example

```
#include <ui_env.hpp>

ExampleFunction(int value)
{
    if (AbsValue(value) < 256)
    {


    }
}
```

attrib

Syntax

```
#include <ui_dsp.hpp>

#if defined(ZIL_CURSES)
#    if defined(SCO_UNIX)
#        define attrib(foreground, background)
             (COLOR_PAIR(foreground « 3 | background))
#    else
#        define attrib (foreground, background) (0)
#    endif
#else
#    define attrib(foreground, background) (((background) « 4) + (foreground))
#endif
```

Portability

This macro is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This macro combines text *foreground* and *background* color values into one value that the operating system can use. attrib is used by the UI_PALETTE structure to describe color and monochrome text attributes.

Example

```
#include <ui_dsp.hpp>

static UI_PALETTE backgroundPalette =
{
    '\260', attrib(BLUE, BLACK), attrib(MONO_DIM, MONO_BLACK),
    PTN_INTERLEAVE_FILL, BLUE, BLUE, BW_WHITE, BW_WHITE, GS_GRAY, GS_GRAY
};

static UI_PALETTE xorPalette =
{
    '\260', attrib(BLUE, BLACK), attrib(MONO_DIM, MONO_BLACK),
    PTN_SOLID_FILL, LIGHTGRAY, LIGHTGRAY, BW_WHITE, BW_WHITE, GS_GRAY, GS_GRAY
};
```

FlagSet

Syntax

include <ui_env.hpp>

#define *FlagSet(flag1,flag2)* ((flag1) & (flag2))

Portability

This macro is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This macro determines if any flags set in *flag2* are set in *flag1.* It does this by AND'ing the two values together. For example, if one argument were a 0 and the other were a 1, the result would be FALSE, since there are no bits in the arguments that overlap. On the other hand, if one argument were 0x0001 and the other were 0x0101, the result would be 0x0001 (TRUE).

Example

```
#include <ui_win.hpp>

void ExampleFunction(UIF_FLAGS flags)
{
    if FlagSet(flags, WOF_NO_ALLOCATE_DATA)
    {
```

```
    }
}
```

FlagsSet

Syntax

```
#include <ui_env.hpp>

#define FlagsSet(flagl, flag2) (((flagl) & (flag2)) = (flag2))
```

Portability

This macro is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This macro is similar to **FlagSet,** except that it checks to see if all the flags set in *flag2* are set in *flagl.* It does this by AND'ing the two values together and comparing the result to *flag2.* If the results are the same, the macro evaluates to TRUE. Otherwise it is FALSE. For example, if one flag were a 0 and the other were a 1, the result would be 0 (FALSE), since there are no flags that overlap. If *flagl* were 0x0100 and *flag2* were OxOFOO, the result would be 0x0100, which is also FALSE, while these flags reversed (i.e., OxOFOO, 0x0100) would result in TRUE.

Both **FlagSet** and **FlagsSet** are used extensively throughout the library when comparing window flags (i.e., WOF_FLAGS and WOAF_FLAGS) or when comparing status flags (i.e., WOS_STATUS). They are also used in window object derived classes when flags are compared (e.g., BTF_ flags in the button class). The Event Manager also uses **FlagSet** and **FlagsSet** with the **UI_EVENT_MANAGER::Get()** function to determine the point of the queue from which the event will be retrieved.

Example

```
#include <ui_win.hpp>

void ExampleFunction (UIF_FLAGS flags)
{
    if FlagsSet(flags, WOF_BORDER | WOF_NON_FIELD_REGION)
    {


    }
}
```

HIWORD

Syntax

include <ui_env.hpp>

define HIWORD*(arg)* (((ULONG)arg » 16) & OxOOOOFFFF)

Portability

This macro is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This macro returns the high word of a 32-bit value.

Example

```
#include <ui_win.hpp>

void ExampleFunction(ZIL_UINT32 value)
{
    ZIL_UINT16 hiWord = HIWORD(value);


}
```

LOWORD

Syntax

include <ui_env.hpp>

define LOWORD(arg) ((ULONG)arg & OxOOOOFFFF)

Portability

This macro is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

This macro returns the low word of a 32-bit value.

Example

```
#include <ui_win.hpp>
void ExampleFunction(ZIL_UINT32 value)
{
    ZIL_UINT16 loWord = LOWORD(value);
}
```

MaxValue

Syntax

include <ui_env.hpp>

#define MaxValue(«rgi, *argl)* (((argl) > (arg2)) ? (argl) : (arg2))

Portability

This macro is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This macro returns the larger of the two values.

Example

```
#include <ui_dsp.hpp>
int UI_REGION_ELEMENT::Overlap(const UI_REGION &tRegion)
{
    return (MaxValue(tRegion.left, region.left) <=
        MinValue(tRegion.right, region.right) &&
        MaxValue(tRegion.top, region.top) <=
        MinValue(tRegion.bottom, region.bottom));
```

MinValue

Syntax

include <ui_env.hpp>

#define MinValue(arg7, *arg2)* (((argl) < (arg2)) ? (argl) : (arg2))

Portability

This macro is available on the following environments:

- DOS Text • DOS Graphics • Windows • OS/2
- Macintosh • OSF/Motif • Curses • NEXTSTEP

Remarks

This macro returns the smaller of the two values.

Example

```
#include <ui_env.hpp>

int UI_REGION_ELEMENT::Overlap(const UI_REGION ktRegion)
{
    return (MaxValue(tRegion.left, region.left) <=
        MinValue(tRegion.right, region.right) &&
        MaxValue(tRegion.top, region.top) <=
        MinValue(tRegion.bottom, region.bottom));
}
```

ZIL_NULLF

Syntax

include <ui_env.hpp>

#define *ZIL_NULLF(*type) ((type)O)

Portability

This macro is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

The *ZIL_NULLF* macro is used to typecast a NULL function pointer. This macro is used when a NULL function pointer is required.

ZIL_NULLH

Syntax

include <ui_env.hpp>

#define *ZIL_NULLH(*type) ((type)O)

Portability

This macro is available on the following environments:

- DOS Text • DOS Graphics • Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

The *ZIL_NULLH* macro is used to typecast a NULL handle pointer. This macro is used when a NULL handle pointer is required.

ZIL_NULLP

Syntax

include <ui_env.hpp>

#define *ZIL_NULLP(*type) ((type *)0)

Portability

This macro is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

The *ZIL_NULLP* macro is used to typecast a type pointer to NULL. This macro is used when a NULL object pointer is required.

ZIL_VOIDF

Syntax

include <ui_env.hpp>

#define *ZIL_VOIDF(*function) (function)
#if defined(__BCPLUSPLUS__) || defined(_TCPLUSPLUS_)
define ZIL_VOIDF(function) ((void *)(function))

Portability

This macro is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

The *ZIL_VOIDF* macro is used to typecast a function pointer according to the requirements of the environment. The definition of *ZIL_VOIDF* varies depending on the environment. This macro is typically used in UI_ITEM arrays.

ZIL_VQ1PP

Syntax

include <ui_env.hpp>

#define *ZIL_VOIDP(*pointer) (pointer)
#if defined(__BCPLUSPLUS__) || defined(_TCPLUSPLUS_)
define *ZIL_VOIDP(*pointer) ((void *)(pointer))

Portability

This macro is available on the following environments:

- DOS Text
- DOS Graphics
- Windows
- OS/2
- Macintosh
- OSF/Motif
- Curses
- NEXTSTEP

Remarks

The *ZIL_VOIDP* macro is used to typecast a data pointer according to the requirements of the environment. The definition of *ZIL_VOIDP* varies depending on the environment.

APPENDIX B - SYSTEM EVENTS

This appendix describes the system events that can be generated in OpenZinc Application Framework. System messages are passed using the UI_EVENT structure, where the system message is contained in *EVENT_TYPE* and any related information is contained in the union portion of the UI_EVENT structure. (For additional information about system event mapping, see the **Event()** member functions associated with window objects.) The following messages (declared in **UI_EVT.HPP)** are recognized within OpenZinc Application Framework:

S_ADD_OBJECT—Causes a new object to be added to the list, *event.data* will point to the new object to be added.

S_ALT_KEY—Causes focus to move from the user region to the pull-down menu or, if the pull-down menu has focus, from the pull-down menu to the current object on the user region of the window.

S_CHANGED—Causes the object to recalculate its position and size. When a window is moved or sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_CHANGE_PAGE—Causes the notebook to "turn" to a new page. The page number that should be turned to is subtracted from S_CHANGE_PAGE and passed as the event type. The first page that was added is page zero. For example, if ten pages were added, and the application needs to turn to page seven, an event with a type of S_CHANGE_PAGE - 6 should be sent to the notebook. Thus, an S_CHANGE_PAGE event by itself will turn to the first page in the notebook.

S_CREATE—Causes the object to create itself. The object will calculate its position and size and, if necessary, will register itself with the operating system. This message is sent by the Window Manager when a window is attached to it to cause the window and all the objects attached to the window to determine their positions.

S_CURRENT—Causes the object to draw itself to appear current. This message is sent by the Window Manager to a window when it becomes current. The window, in turn, passes this message to the object on the window that is current.

S_DEINITIALIZE—Informs the object that it is about to be removed from the application and that it should deinitialize any information. The Window Manager

sends this message to a window when the window is subtracted from the Window Manager. The window, in turn, relays the message to all objects attached to it.

S_DISPLAY_ACTIVE—Causes the object to draw itself to appear active. An active object is one that is on the active (i.e., current) window. Most objects do not display differently whether they are active or inactive. An active object should not be confused with a current object. An object is active if it is on the active window. However, it may not be the current object on the window.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed by the event overlaps the region of the object.

S_DISPLAY_INACTIVE—Causes the object to draw itself to appear inactive. An inactive object is one that is not on the active (i.e., current) window. Most objects do not display differently whether they are inactive or active.

The region that needs to be redisplayed is passed in the UI_REGION portion of the UI_EVENT structure when this message is sent. The object only needs to redisplay when the region passed with the event overlaps the region of the object.

S_DRAG_COPY_OBJECT—Indicates the user is dragging the object to copy it.

S_DRAG_MOVE_OBJECT—Indicates the user is dragging the object to move it.

S_DROP_COPY_OBJECT—Indicates the user dropped an object to copy it to this object.

S_DROP_MOVE_OBJECT—Indicates the user dropped an object to move it to this object.

S_HIDE_DEFAULT—Causes the object to draw as a normal button when it has been drawing as the default button. The default button has a thick border. This message is sent by another object when the object wishes to appear as the default.

S_HSCROLL—Causes the object to scroll horizontally, *event.scrolldelta* indicates how far to scroll.

S_HSCROLL_CHECK—Causes the object to scroll the current item into view if it is not currently visible.

S_HSCROLL_SET—Sets the scroll information for the horizontal scroll bar or slider. The thumb button location will be updated to reflect the values, *event.scroll* will contain the new scroll information.

S_INITIALIZE—Causes the object to initialize any necessary information that may require a knowledge of its parent or siblings. When a window is added to the Window Manager, the Window Manager sends this message to cause the window and all the objects attached to the window to initialize themselves.

S_**MAXIMIZE—**Causes the window to be maximized so that it is as large as allowed. If the window is added to the Window Manager it will be the size of the screen. If the window is an MDI child, it will be as large as its parent's user region. If the window is already in a maximized state, S_MAXIMIZE causes it to return to its original size.

S_MDICHILD_EVENT + S_CLOSE—Causes the current MDI child to be closed. The MDI parent will subtract the current MDI child and, if the MDI child does not have the WOAF_NO_DESTROY flag set, will delete the child.

S_MDICHILD_EVENT + S_MAXIMIZE—Causes the current MDI child window to be maximized so that it is as large as its parent's user region. If the window is already in a maximized state, S_MDI_CHILD_EVENT + S_MAXIMIZE causes it to return to its original size.

S_MDICHILD_EVENT + S_MINIMIZE—Causes the current MDI child window to be minimized. If the window has a minimize icon, it will be displayed. If the window is already in a minimized state, S_MDI_CHILD_EVENT + S_MINIMIZE causes it to return to its original size.

S_MDICHILD_EVENT + S.RESTORE—Causes the current MDI child window to return to its normal size if the window was in a maximized or minimized state.

S_MINIMIZE—Causes the window to be minimized. If the window has a minimize icon, it will be displayed. If the window is already in a minimized state, S_MINIMIZE causes it to return to its original size.

S_MOVE—Causes the object to update its location. The distance to move is contained in the *position* field of UI_EVENT. For example, an *event.position.line* of -10 and an *event.position.column* of 15 moves the object 10 lines up and 15 columns to the right.

S_NON_CURRENT—Indicates that the object has just become non-current. This message is received when the user moves to another field or window.

S_REDISPLAY—Causes the object to redraw.

S_REGION_DEFINE—Causes the object to reserve a region of the screen in which it will display.

S_REGISTER_OBJECT—Causes the object to register itself with the operating system.

S_RESET_DISPLAY—Changes the display to a different resolution, *event.data* should point to the new display class to be used. If *event.data* is NULL, then a text mode display will be created. This event is specific to DOS and must be placed on the event queue by the programmer. The library will never generate this event.

S_RESTORE—Causes the window to return to its normal size if the window was in a maximized or minimized state.

S_SCROLLRANGE—Calculates the scroll region for the window.

S_SET_DATA—Causes the record to update the data in its fields, *event.rawCode* contains the record number and *event.data* contains the data for the record. If the UIW_TABLE_RECORD processes this message (i.e., the table record is not a derived table record) it will call the user function, if one exists, with a ccode of S_SET_DATA. *event* is sent to the user function.

S_SHOW_DEFAULT—Causes the object to draw as the default button. The default button has a thick border. This message is sent when another button has been current and displaying as the default button but is no longer current.

S_SIZE—Causes the object to recalculate its position and size. When a window is sized, the objects on the window will need to recalculate their positions. This message informs an object that it has changed and that it should update itself.

S_SUBTRACT_OBJECT—Causes an object to be subtracted from the list. *event.data* will point to the object to be subtracted.

S_VERIFY_STATUS—Causes the object to correlate its state (e.g., selected or not selected) with the operating system.

S_VSCROLL—Causes the table to scroll vertically, *event.scroll.delta* indicates how far to scroll.

S_VSCROLL_CHECK—Causes the list to scroll the current item into view if it is not currently visible.

S_VSCROLL_SET—Sets the scroll information for the vertical scroll bar or slider. The thumb button location will be updated to reflect the values, *event.scroll* will contain the new scroll information.

APPENDIX C - LOGICAL EVENTS

This appendix describes the logical events that can be interpreted or generated in OpenZinc Application Framework. Logical events are passed using the UI_EVENT structure, where the logical message can either be contained directly in *EVENT_TYPE* or interpreted from *event.rawCode* using **MapEvent()**. Any related information is contained in the union portion of the UI_EVENT structure. (For additional information about logical event mapping, see the **Event()** member functions associated with window objects.)

L_BACKSPACE—Causes the first editable character to the left of the cursor position to be deleted and moves the cursor to that position. This message is interpreted from a keyboard event.

L_BEGIN_MARK—Indicates the marking process is beginning.

L_BEGIN_SELECT—Indicates that the end-user began the selection of the object by pressing the mouse button down while on the object.

L_BOL—Causes the cursor to be moved to the first editable character in the string. This message is interpreted from a keyboard event.

L_BOTTOM—Scrolls the list to the last page and makes the last item in the list current. This message is interpreted from a keyboard event.

L_CANCEL—Causes the current action to be cancelled.

L_CONTINUE_MARK—Indicates the marking process is continuing.

L_CONTINUE_SELECT—Indicates that the end-user previously clicked down on the object with the mouse and is now continuing to hold the mouse button down while on the object.

L_COPY_MARK—Causes the marked region to be copied into the global paste buffer. This message is interpreted from a keyboard event.

L_CUT—Cuts the marked portion of the string. The cut region is stored in the global paste buffer. This message is interpreted from a keyboard event.

L_DELETE—Causes the marked characters, if any, or the character at the current cursor position to be deleted.

L_DELETE_EOL—Causes all editable characters from the current cursor position to the end of the field to be deleted. This message is interpreted from a keyboard event.

L_DELETE_WORD—Causes the word at the cursor position to be deleted, along with any trailing spaces. This message is interpreted from a keyboard event.

L_DOUBLE_CLICK—Indicates that the end-user double-clicked on the object with the mouse.

L_DOWN—Moves the focus down one object or decrements a value, depending on the type of object. This message is interpreted from a keyboard event.

L_END_MARK—Indicates that the end-user has finished marking text in the string. This message is interpreted from a mouse event.

L_END_SELECT—Indicates that the selection process, initiated with the L_-BEGIN_SELECT message, is complete. For example, the end-user has pressed and released the mouse button.

L_EOL—Causes the cursor to be moved to the last editable character in the string. This message is interpreted from a keyboard event.

L_FIRST—Causes the first object in the list to be made current.

L_HELP—Causes the help system to be displayed. The window passes this message to the current object to let it display its help. If the current object does not process the message, the window's help context will be displayed.

L_INSERT_TOGGLE—Toggles the insert mode. This message is interpreted from a keyboard event.

L_LAST—Causes the last object in the list to be made current.

L_LEFT—Moves the focus left or decrements a value, depending on the type of object. This message is interpreted from a keyboard event.

L_MARK—Turns the mark feature on or off. This message is interpreted from a keyboard event.

L_MARK_BOL—Marks the text from the current cursor position to the beginning of the current line and places the cursor at the beginning of the line. This message is interpreted from a keyboard event.

L_MARK_DOWN—Causes the cursor to move down one line in the text buffer. Where possible, the cursor position stays at the same horizontal character offset. The text between the starting cursor position and the ending cursor position will be marked. This message is interpreted from a keyboard event.

L_MARK_EOL—Marks the text from the current cursor position to the end of the current line and places the cursor at the end of the line. This message is interpreted from a keyboard event.

L_MARK_LEFT—Moves the cursor to the left one character, marking the character. This message is interpreted from a keyboard event.

L_MARK_PGDN—Causes the text field to scroll down one page. The text between the starting cursor position and the ending cursor position will be marked. This message is interpreted from a keyboard event.

L_MARK_PGUP—Causes the text field to scroll up one page. The text between the starting cursor position and the ending cursor position will be marked. This message is interpreted from a keyboard event.

L_MARK_RIGHT—Moves the cursor to the right one character, marking the character. This message is interpreted from a keyboard event.

L_MARK_UP—Causes the cursor to move up one line in the text buffer. Where possible, the cursor position stays at the same horizontal character offset. The text between the starting cursor position and the ending cursor position will be marked. This message is interpreted from a keyboard event.

L_MARK_WORD_LEFT—Causes the cursor position to be moved to the beginning of the current word or, if the cursor is at the beginning of the current word, to the beginning of the next word to the left of the current cursor position. The text between the starting cursor position and the ending cursor position will be marked. This message is interpreted from a keyboard event.

L_MARK_WORD_RIGHT—Causes the cursor to move to the beginning of the next word to the right of the current cursor position. The text between the starting cursor position and the ending cursor position will be marked. This message is interpreted from a keyboard event.

L_MAXIMIZE—Causes the window to be maximized so that it is as large as allowed. If the window is added to the Window Manager it will be the size of the screen. If the window is an MDI child, it will be as large as its parent's user region.

If the window is already in a maximized state, L_MAXIMIZE causes it to return to its original size.

L_MDICHILD_EVENT + L_MOVE—Causes the current MDI child window to go into "move mode." If, for example, the end-user selects the "Move" option from the system menu, the window can then be moved using the arrow keys.

L_MDICHILD_EVENT + L_NEXT_WINDOW—Makes the next MDI child the current MDI child.

L_MDICHILD_EVENT + L_SIZE—Causes the current MDI child window to go into "size mode." If, for example, the end-user selects the "Size" option from the system menu, the window can then be sized using the arrow keys.

L_MINIMIZE—Causes the window to be minimized. If the window has a minimize icon, it will be displayed. If the window is already in a minimized state, L_MINIMIZE causes it to return to its original size.

L_MOVE—Causes the window to go into "move mode." If, for example, the end-user selects the "Move" option from the system menu, the window can then be moved using the arrow keys.

L_NEXT—Causes the next selectable object in the list of window objects to become current. If the last field on the window is current, the first object will become current unless the WNF_NO_WRAP flag is set. This message is interpreted from a keyboard event.

L_PASTE—Causes the contents of the paste buffer to be placed in the field at the current cursor position.

L_PGDN—Causes the list to scroll down a page. This message is interpreted from a keyboard event.

L_PGUP—Causes the list to scroll up a page. This message is interpreted from a keyboard event.

L_PREVIOUS—Causes the previous selectable object in the list of window objects to become current. If the first field on the window is current, the last object will become current unless the WNF_NO_WRAP flag is set. This message is interpreted from a keyboard event.

L_RESTORE—Causes the window to return to its normal size if the window was in a maximized or minimized state.

L_RIGHT—Moves the focus right or increments a value, depending on the type of object. This message is interpreted from a keyboard event.

L_SELECT—Indicates that the object has been selected. The selection may be the result of a mouse click or a keyboard action.

L_SIZE—Causes the window to go into "size mode." If, for example, the end-user selects the "Size" option from the system menu, the window can then be sized using the arrow keys.

L_TOP—Scrolls the list to the first page and makes the first item in the list current. This message is interpreted from a keyboard event.

L_UP—Moves the focus up one object or increments a value, depending on the type of object. This message is interpreted from a keyboard event.

L_VIEW—Indicates that the mouse is being moved over the object. This message allows the object to alter the mouse image.

L_WORD_LEFT—Causes the cursor position to be moved to the beginning of the current word or, if the cursor is at the beginning of the current word, to the beginning of the next word to the left of the current cursor position. This message is interpreted from a keyboard event.

L_WORD_RIGHT—Causes the cursor to move to the beginning of the next word to the right of the current cursor position.

APPENDIX D - CLASS IDENTIFIERS

This appendix contains a list of fixed constant values that are used as class identifiers. The definition of these constants is contained in **UI_GEN.HPP.**

General

The following identifications are general to OpenZinc Application Framework:

ID_DISPLAY—Identification for the UI_DISPLAY class.

ID_EVENT_MANAGER—Identification for the UI_EVENT_MANAGER class.

ID_WINDOW_MANAGER—Identification for the UI_WINDOW_MANAGER class.

ID_END—Identification that indicates the end of an array (used by **MapEvent( )** and **MapPalette()).**

ID_SCREEN—Identification for the screen background.

ID_DIRECT—Identification for the screen directly, including any objects on the screen.

Window Objects

The following identifications apply to window objects:

ID_ATTACHMENT	UI_ATTACHMENT class
ID_BIGNUM	UIW_BIGNUM class
ID_BORDER	UIW_BORDER class
ID_BUTTON	UIW_BUTTON class
ID_CHECK_BOX	check box
ID_COMBO_BOX	UIW_COMBO_BOX class

ID_CONSTRAINT	UI_CONSTRAINT class
ID DATE	UIW_DATE class
ID DIMENSION CONSTRAINT	UI_DIMENSION_CONSTRAINT class
ID_FORMATTED_STRING	UIW_FORMATTED_STRING class
ID_GEOMETRY_MANAGER	UI_GEOMETRY_MANAGER class
ID_GROUP	UIW_GROUP class
ID_HZ_LIST	UIW_HZ_LIST class
ID_ICON	UIW_ICON class
ID_INTEGER	UIW_INTEGER class
ID_LIST	Any list type
ID_LIST_ITEM	Any object in a list
ID_MAXIMIZE_BUTTON	UIW_MAXIMIZE_BUTTON class
ID_MENU	Used to tie the menu classes
ID_MENU_ITEM	Used to tie the menu item classes
ID_MINIMIZE_BUTTON	UIW_MINIMIZE_BUTTON class
ID_NOTEBOOK	UIW_NOTEBOOK class
ID_NUMBER	Used to tie the number classes
ID_POP_UP_MENU	UIW_POP_UP_MENU class
ID_POP_UP_ITEM	UIW_POP_UP_ITEM class
ID_PROMPT	UIW_PROMPT class
ID_PULL_DOWN_MENU	UIW_PULL_DOWN_MENU class
ID PULL DOWN ITEM	UIW PULL DOWN ITEM class

ID_RADIO_BUTTON	radio button
ID_REAL	UIW_REAL class
ID_RELATIVE_CONSTRAINT	UI_RELATIVE_CONSTRAINT class
ID_SCROLL_BAR	UIW_SCROLL_BAR class
ID_SPIN_CONTROL	UIW_SPIN_CONTROL class
ID_STATUS_BAR	UIW_STATUS_BAR class
ID_STATUS_ITEM	An object on a status bar
ID_STRING	UIW_STRING class
ID_SYSTEM_BUTTON	UIW_SYSTEM_BUTTON class
ID_TABLE	UIW_TABLE class
ID_TABLE_HEADER	UIW_TABLE_HEADER class
ID_TABLE_RECORD	UIW_TABLE_RECORD class
ID_TEXT	UIW_TEXT class
ID_TIME	UIW_TIME class
ID_TITLE	UIW_TITLE class
ID_TOOL_BAR	UIW_TOOL_BAR class
ID_VT_LIST	UIW_VT_LIST class
ID_WINDOW	UIW_WINDOW class
ID_WINDOW_OBJECT	UI_WINDOW_OBJECT class

Shadowing

The following identifications are used to determine the color of a shaded object (e.g., border, button):

ID_OUTLINE	Outline of the object
ID_WHITE_SHADOW	Top-left shadow (normal)
ID_LIGHT_SHADOW	Bottom-right shadow (normal)
ID_DARK_SHADOW	Top-left shadow (depressed)
ID_BLACK_SHADOW	Bottom-right shadow (depressed)

APPENDIX E - OpenZinc OBJECT STORAGE

This appendix describes the file layout for <*file*>.**DAT** files. These files are created by the Interactive Design Tool whenever the *file, Save" or *file, save As" option is selected.

File Information

Each **.DAT** file contains all objects created and saved by OpenZinc Designer. Each file is organized in the following manner:

OpenZinc Signature
Revision Number
UIW_WINDOW directory
- contains definitions for UIW_WINDOW as well as the window's sub-objects.

UI_BITMAP directory
- contains bitmap data (i.e., name, height, width, bitmap array).

UI_ICON directory
- contains icon data (i.e., name, text, icon array).

UI_HELP directory
- contains help contexts (i.e., help context, title, message).

UI_HPP directory
- contains information used to create the **.HPP** file.

UI_CPP directory
- contains entries to connect window objects with their corresponding userFunction (specified in OpenZinc Designer).

ZIL_INTERNATIONAL directory
- contains locale and language translations. Not all **.DAT** files will have this directory.

OpenZinc signature is stored by the OpenZinc_SIGNATURE structure (defined in **STORE.CPP):**

```
struct OpenZinc_SIGNATURE
{
    char copyrightNotice[64];
    ZIL_UINT8 majorVersion;
    ZIL_UINT8 minorVersion;
    ZIL_UINT16 magicNumber;
};
```

UI_ATTACHMENT

The UI_ATTACHMENT class stores the following member variables after calling **UI_CONSTRAINT::Store():**

reference->numberID, if the attachment is tied to an object.
refObjectID
atcFlags
offset

UI_CONSTRAINT

The UI_CONSTRAINT class stores the following member variable:

object->numberID, if the constraint is tied to an object.

UI_DIMENSION_CONSTRAINT

The UI_DIMENSION_CONSTRAINT class stores the following member variables after calling **UI_CONSTRAINT::Store( ):**

dncFlags
maximum
minimum

UI_GEOMETRY_MANAGER

The UI_GEOMETRY_MANAGER class stores the following member variables after calling **UI_WINDOW_OBJECT::Store( ):**

The number of constraints attached to the geometry manager.
The *searchID* of the constraint.
The constraint.

UI_RELATIVE_CONSTRAINT

The UI_RELATIVE_CONSTRAINT class stores the following member variables after calling **ULCONSTRAINT::Store():**

rlcFlags

numerator
denominator
offset

UI_WINDOW_OBJECT

The UI_WINDOW_OBJECT class stores the following member variables:

numberID
stringID
woFlags
woAdvancedFlags
left
top
right
bottom
helpContext
userFlags
userStatus
userObjectName
userFunctionName

UIW_BIGNUM

The bignum class stores the following member variables, after calling UIW_STRING::-Store():

nmFlags
range

NOTE: The bignum value is saved by storing the string representation of the bignum when UIW_STRING::Store() is called.

UIW_BORDER

The border is stored as an attribute of UIW_WINDOW.

UIW_BUTTON

The UIW_BUTTON class stores the following member variables, after calling **UI_-WINDOW_OB**JECT::**Store():**

btFlags
value
depth
text
bitmapName

NOTE: If a bitmap is associated with the button, it is stored in the UI_BITMAP directory.

UIW_COMBO_BOX

The UIW_COMBO_BOX class does not store any member variables. It calls **UI_-WINDOW_OB**JECT::**Store()** then stores its associated list by calling the list's **Store( )** function.

UIW_DATE

The UIW_DATE class stores the following member variables, after calling **UIW_-STRING::Store():**

dtFlags
range

NOTE: The date value is saved by storing the string representation of the date when **UIW_STRING::Store()** is called.

UIW_FORMATTED_STRING

The UIW_FORMATTED_STRING class stores the following member variables, after calling **UIW_STRING::Store( ):**

compressedText
editMask
deleteText

UIW_GROUP

The UIW_GROUP class stores the following member variable, after calling **UIW_-WINDOW::Store():**

text

UIW_HZ_LIST

The UIW_HZ_LIST class stores the following member variables, after calling **UIW_-WINDOW::Store():**

cellWidth
cellHeight

UIW_ICON

The UIW_ICON class stores the following member variables, after calling **UI_-WINDOW_OBJECT::Store():**

icFlags
title
iconName
iconWidth
iconHeight
iconArray

NOTE: If a bitmap is associated with the button, it is stored in the UI_BITMAP directory.

UIW_INTEGER

The UIW_INTEGER class stores the following member variables, after calling **UIW_-STRING::Store():**

nmFlags
range

NOTE: The integer value is saved by storing the string representation of the integer when **UIW_STRING::Store()** is called.

UIW_MAXIMIZE_BUTTON

The UIW_MAXIMIZE_BUTTON class only stores its *searchID,* which is ID_-MAXIMIZE_BUTTON.

UIW_MINIMIZE_BUTTON

The UIW_MINIMIZE_BUTTON class only stores its *searchID,* which is ID_-MINIMIZE_BUTTON.

UIW_POP_UP_ITEM

The UIW_POP_UP_ITEM class stores the following member variable, after calling **UIW_BUTTON::Store()**:

mniFlags

NOTE: The pop-up item also stores its associated menu (if any) by calling the menu's **Store(**) function.

UIW_POP_UP_MENU

The UIW_POP_UP_MENU class does not store any member variables, it only calls **UIW_WINDOW::Store().**

UIW_PROMPT

The UIW_PROMPT class stores the following member variable, after calling UI_-**WINDOW_OBJECT::Store():**

text

UIW_PULL_DOWN_ITEM

The UIW_PULL_DOWN_ITEM class does not store any member variables. It calls **UIW_BUTTON::Store**() then stores its associated menu (if any) by calling the menu's **Store()** function.

UIW_PULL_DOWN_MENU

The UIW_POP_UP_MENU class does not store any member variables, it only calls **UIW_WINDOW::Store().**

UIW_REAL

The UIW_REAL class stores the following member variables, after calling **UIW_-STRING::Store():**

nmFlags
range

NOTE: The real value is saved by storing the string representation of the real when **UIW_STRING::Store()** is called.

UIW_SCROLL_BAR

The **UIW_SCROLL_BAR** class stores the following member variables, after calling **UI_-WINDOW_OBJECT::Store():**

sbFlags
minimum
maximum
current

UIW_SPIN_CONTROL

The UIW_SPIN_CONTROL class stores the following member variables, after calling **UI_WINDOW_OBJECT::Store():**

The *searchID* of the object being controlled.
fieldObject (the object being controlled)
wnFlags
delta

UIW_STATUS_BAR

The UIW_STATUS_BAR class stores the following member variable, after calling **UIW_-WINDOW::Store():**

height

UIW_STRING

The UIW_STRING class stores the following member variables, after calling **UI_-WINDOW_OBJECT::Store():**

stFlags
maxLength
text

UIW_SYSTE M_BUTTO N

The UIW_SYSTEM_BUTTON class stores the following member variable:

syFlags.

If the system button is not a generic system button its menu is also stored.

UIW_TABLE

The UIW_TABLE class stores the following member variable, after calling **UIW_-WINDOW::Store():**

tblFlags

UIW_TABLE_HEADER

The UIW_TABLE_HEADER class stores the following member variable, after calling **UIW_TABLE::Store():**

thFlags

UIW_TABLE_RECORD

The UIW_TABLE_RECORD class does not store any member variables, it only calls **UIW_WINDOW::Store( ).**

UIW_TEXT

The UIW_TEXT class stores the following member variables, after calling **UI_-WINDOW_OBJECT::Store():**

maxLength
text
noOfObjects
_value (This value is stored for each of the text object's support objects.)
object (This is a support object.)
wnFlags

UIW_TIME

The UIW_TIME class stores the following member variables, after calling **UIW_-STRING::Store():**

tmFlags
range

NOTE: The time value is saved by storing the string representation of the time when **UIW_STRING::Store()** is called.

UIW_TITLE

The UIW_TITLE class stores the following member variable:

text

UIW_TOOL_BAR

The UIW_TOOL_BAR class does not store any member variables, it only calls **UIW_-WINDOW::Store().**

UIW_VT_LIST

The UIW_VT_LIST class does not store any member variables, it only calls **UIW_-WINDOW::Store().**

UIW_WINDOW

The UIW_WINDOW class:

1—Checks for a valid directory (i.e., in the file) and disk file. If the file does not exist (i.e., a new file), the file is created and the following variables are stored:

miniNumeratorX
miniDenominatorX
miniNumeratorY
miniDenominatorY

2—The window and each of the sub-objects are stored. First **UI_-WINDOW_OBJECT::Store()** is called to store the window and then the following member variables are stored:

noOfObjects (i.e., number of objects attached to the window)

Support Objects (The *object->searchID* is stored for each object and then the **object->Store()** is called. This is done for all of the window's support objects.)

Regular Objects (The *object->searchID* is stored for each object and then the **object->Store()** is called. This is done for all of the window's objects that are not in the support list.)

wnFlags
compareFunctionName

3—Write out the header information. Header information (used to re-construct the object) is stored for each object in the window.

4—User and compare function names are stored together with a logical link to the objects to which they are attached.

ZAF_DIALOG_WINDOW

The ZAF_DIALOG_WINDOW class does not store any member variables, it only calls UIW_WINDOW::Store().

APPENDIX F – CHARACTER SETS

ISO8859-1 Character Set — Decimal

	0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200	210	220	230	240	250
0		◙	¶	▲	(	2	<	F	P	Z	d	n	x	█	┘	╙		ª	´	¾	È	Ò	Ü	æ	ð	ú
1	☺	♂	§	▼	)	3	=	G	Q	[	e	o	y	│	┌	╨	¡	«	µ	¿	É	Ó	Ý	ç	ñ	û
2	☻	♀	▬	Space	*	4	>	H	R	\	f	p	z	┤		╥	¢	¬	¶	À	Ê	Ô	Þ	è	ò	ü
3	♥	♪	↨	!	+	5	?	I	S	]	g	q	{	┐		╟	£	-	·	Á	Ë	Õ	ß	é	ó	ý
4	♦	♫	↑	"	,	6	@	J	T	^	h	r	|	└	▀	═	¤	®	¸	Â	Ì	Ö	à	ê	ô	þ
5	♣	☼	↓	#	-	7	A	K	U	_	i	s	}	┴	▄	╬	¥	¯	¹	Ã	Í	×	á	ë	õ	ÿ
6	♠	►	→	$	.	8	B	L	V	`	j	t	~	┬	█	╝	¦	°	º	Ä	Î	Ø	â	ì	ö	
7	•	◄	←	%	/	9	C	M	W	a	k	u	⌂	├	║	╔	§	±	»	Å	Ï	Ù	ã	í	÷	
8	◘	↕	∟	&	0	:	D	N	X	b	l	v	▒	─	╣	■	¨	²	¼	Æ	Ð	Ú	ä	î	ø	
9	○	‼	↔	'	1	;	E	O	Y	c	m	w	█	┼	╗		©	³	½	Ç	Ñ	Û	å	ï	ù	

Characters in shaded boxes are not true ISO characters but are available in Zinc's DOS text mode only

IBM Code Page 850 — Decimal

	0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200	210	220	230	240	250
0		◙	¶	▲	(	2	<	F	P	Z	d	n	x	é	î	û	á	¬	┤	¥	╚	Ê	▄	µ	-	·
1	☺	♂	§	▼	)	3	=	G	Q	[	e	o	y	â	ì	ù	í	½	Á	┐	╔	Ë	¦	þ	±	¹
2	☻	♀	▬	SPACE	*	4	>	H	R	\	f	p	z	ä	Ä	ÿ	ó	¼	Â	└	╩	È	Ì	Þ	‗	³
3	♥	♪	↨	!	+	5	?	I	S	]	g	q	{	à	Å	Ö	ú	¡	À	┴	╦	ı	▀	Ú	¾	²
4	♦	♫	↑	"	,	6	@	J	T	^	h	r	¦	å	É	Ü	ñ	«	©	┬	╠	Í	Ó	Û	¶	■
5	♣	☼	↓	#	-	7	A	K	U	_	i	s	}	ç	æ	ø	Ñ	»	╣	├	═	Î	ß	Ù	§	
6	♠	►	→	$	.	8	B	L	V	`	j	t	~	ê	Æ	£	ª	░	║	─	╬	Ï	Ô	ý	÷	
7	•	◄	←	%	/	9	C	M	W	a	k	u	⌂	ë	ô	Ø	º	▒	╗	┼	¤	┘	Ò	Ý	¸	
8	◘	↕	∟	&	0	:	D	N	X	b	l	v	Ç	è	ö	×	¿	▓	╝	ã	ð	┌	õ	¯	°	
9	○	‼	↔	'	1	;	E	O	Y	c	m	w	ü	ï	ò	ƒ	®	│	¢	Ã	Ð	█	Õ	´	¨	

IBM Code Page 437 — Decimal

<table>
<tr><th></th><th>0</th><th>10</th><th>20</th><th>30</th><th>40</th><th>50</th><th>60</th><th>70</th><th>80</th><th>90</th><th>100</th><th>110</th><th>120</th><th>130</th><th>140</th><th>150</th><th>160</th><th>170</th><th>180</th><th>190</th><th>200</th><th>210</th><th>220</th><th>230</th><th>240</th><th>250</th></tr>
<tr><td>0</td><td></td><td>◙</td><td>¶</td><td>▲</td><td>(</td><td>2</td><td><</td><td>F</td><td>P</td><td>Z</td><td>d</td><td>n</td><td>x</td><td>é</td><td>î</td><td>û</td><td>á</td><td>¬</td><td>┤</td><td>╛</td><td>╚</td><td>╥</td><td>▄</td><td>µ</td><td>≡</td><td>·</td></tr>
<tr><td>1</td><td>☺</td><td>♂</td><td>§</td><td>▼</td><td>)</td><td>3</td><td>=</td><td>G</td><td>Q</td><td>[</td><td>e</td><td>o</td><td>y</td><td>â</td><td>ì</td><td>ù</td><td>í</td><td>½</td><td>╡</td><td>┐</td><td>╔</td><td>╙</td><td>▌</td><td>τ</td><td>±</td><td>√</td></tr>
<tr><td>2</td><td>☻</td><td>♀</td><td>▬</td><td>SPACE</td><td>*</td><td>4</td><td>></td><td>H</td><td>R</td><td>\</td><td>f</td><td>p</td><td>z</td><td>ä</td><td>Ä</td><td>ij</td><td>ó</td><td>¼</td><td>╢</td><td>└</td><td>╩</td><td>╘</td><td>▐</td><td>Φ</td><td>≥</td><td>ⁿ</td></tr>
<tr><td>3</td><td>♥</td><td>♪</td><td>↨</td><td>!</td><td>+</td><td>5</td><td>?</td><td>I</td><td>S</td><td>]</td><td>g</td><td>q</td><td>{</td><td>à</td><td>Å</td><td>Ö</td><td>ú</td><td>¡</td><td>╖</td><td>┴</td><td>╦</td><td>╒</td><td>▀</td><td>Θ</td><td>≤</td><td>²</td></tr>
<tr><td>4</td><td>♦</td><td>♫</td><td>↑</td><td>"</td><td>,</td><td>6</td><td>@</td><td>J</td><td>T</td><td>^</td><td>h</td><td>r</td><td>|</td><td>å</td><td>É</td><td>Ü</td><td>ñ</td><td>«</td><td>╕</td><td>┬</td><td>╠</td><td>╓</td><td>α</td><td>Ω</td><td>⌠</td><td>■</td></tr>
<tr><td>5</td><td>♣</td><td>☼</td><td>↓</td><td>#</td><td>-</td><td>7</td><td>A</td><td>K</td><td>U</td><td>_</td><td>i</td><td>s</td><td>}</td><td>ç</td><td>æ</td><td>¢</td><td>Ñ</td><td>»</td><td>╣</td><td>├</td><td>═</td><td>╫</td><td>ß</td><td>δ</td><td>⌡</td><td></td></tr>
<tr><td>6</td><td>♠</td><td>►</td><td>→</td><td>$</td><td>.</td><td>8</td><td>B</td><td>L</td><td>V</td><td>`</td><td>j</td><td>t</td><td>~</td><td>ê</td><td>Æ</td><td>£</td><td>ª</td><td>░</td><td>║</td><td>─</td><td>╬</td><td>╪</td><td>Γ</td><td>∞</td><td>÷</td><td></td></tr>
<tr><td>7</td><td>•</td><td>◄</td><td>←</td><td>%</td><td>/</td><td>9</td><td>C</td><td>M</td><td>W</td><td>a</td><td>k</td><td>u</td><td>⌂</td><td>ë</td><td>ô</td><td>¥</td><td>º</td><td>▒</td><td>╗</td><td>┼</td><td>╧</td><td>┘</td><td>π</td><td>φ</td><td>≈</td><td></td></tr>
<tr><td>8</td><td>◘</td><td>↕</td><td>∟</td><td>&</td><td>0</td><td>:</td><td>D</td><td>N</td><td>X</td><td>b</td><td>l</td><td>v</td><td>Ç</td><td>è</td><td>ö</td><td>₧</td><td>¿</td><td>▓</td><td>╝</td><td>╞</td><td>╨</td><td>┌</td><td>Σ</td><td>ε</td><td>°</td><td></td></tr>
<tr><td>9</td><td>○</td><td>‼</td><td>↔</td><td>'</td><td>1</td><td>;</td><td>E</td><td>O</td><td>Y</td><td>c</td><td>m</td><td>w</td><td>ü</td><td>ï</td><td>ò</td><td>ƒ</td><td>⌐</td><td>│</td><td>╜</td><td>╟</td><td>╤</td><td>█</td><td>σ</td><td>∩</td><td>∙</td><td></td></tr>
</table>

APPENDIX G - ISO COUNTRY CODES

This appendix lists the ISO country codes. OpenZinc will maintain compatibility with the ISO definitions as they are updated or, in certain cases, before they are officially adopted if it is evident that a proposed standard will be adopted. Please be aware that the inclusion of a country code in this table does not imply support for that country code by OpenZinc Application Framework. This table is the complete ISO table.

Country/Locale Codes

These codes are used by OpenZinc for identifying a particular country, or, if necessary, a locale within a country. The locale identified by these codes will affect the formatting of dates and times, and the display of symbols (e.g., monetary symbols). The codes are from the IS03166 standard.

Country	Code
AFGHANISTAN	AF
ALBANIA	AL
ALGERIA	DZ
AMERICAN SAMOA	AS
ANDORRA	AD
ANGOLA	AO
ANGUILLA	AI
ANTARCTICA	AQ
ANTIGUA AND BARBUDA	AG
ARGENTINA	AR
ARMENIA	AM
ARUBA	AW
AUSTRALIA	AU
AUSTRIA	AT
AZERBAIJAN	AZ
BAHAMAS	BS
BAHRAIN	BH
BANGLADESH	BD

Country	Code
BARBADOS	BB
BELARUS	BY
BELGIUM	BE
BELIZE	BZ
BENIN	BJ
BERMUDA	BM
BHUTAN	BT
BOLIVIA	BO
BOSNIA AND HERZEGOWINA	BA
BOTSWANA	BW
BOUVET ISLAND	BV
BRAZIL	BR
BRITISH INDIAN OCEAN TERRITORY	IO
BRUNEI DARUSSALAM	BN
BULGARIA	BG
BURKINA FASO	BF
BURUNDI	BI
BYELORUSSIAN SSR	BY
CAMBODIA	KH
CAMEROON	CM
CANADA	CA
CAPE VERDE	CV
CAYMAN ISLANDS	KY
CENTRAL AFRICAN REPUBLIC	CF
CHAD	TD
CHILE	CL
CHINA	CN
CHRISTMAS ISLAND	CX
COCOS (KEELING) ISLANDS	CC
COLOMBIA	CO
COMOROS	KM
CONGO	CG
COOK ISLANDS	CK
COSTA RICA	CR

Country	Code
COTE D'IVOIRE	CI
CROATIA (local name: Hrvatska)	HR
CUBA	CU
CYPRUS	CY
CZECH REPUBLIC	CZ
DENMARK	DK
DJIBOUTI	DJ
DOMINICA	DM
DOMINICAN REPUBLIC	DO
EAST TIMOR	TP
ECUADOR	EC
EGYPT	EG
EL SALVADOR	SV
EQUATORIAL GUINEA	GQ
ESTONIA	EE
ETHIOPIA	ET
FALKLAND ISLANDS (MALVINAS)	FK
FAROE ISLANDS	FO
FIJI	FJ
FINLAND	FI
FRANCE	FR
FRANCE, METROPOLITAN	FX
FRENCH GUIANA	GF
FRENCH POLYNESIA	PF
FRENCH SOUTHERN TERRITORIES	TF
GABON	GA
GAMBIA	GM
GEORGIA	GE
GERMANY	DE
GHANA	GH
GIBRALTAR	GI
GREECE	GR
GREENLAND	GL
GRENADA	GD

Country	Code
GUADELOUPE	GP
GUAM	GU
GUATEMALA	GT
GUINEA	GN
GUINEA-BISSAU	GW
GUYANA	GY
HAITI	HT
HEARD AND MC DONALD ISLANDS	HM
HONDURAS	HN
HONG KONG	HK
HUNGARY	HU
ICELAND	IS
INDIA	IN
INDONESIA	ID
IRAN (ISLAMIC REPUBLIC OF)	IR
IRAQ	IQ
IRELAND	IE
ISRAEL	IL
ITALY	IT
JAMAICA	JM
JAPAN	JP
JORDAN	JO
KAZAKHSTAN	KZ
KENYA	KE
KIRIBATI	KI
KOREA, DEMOCRATIC PEOPLE'S REPUBLIC OF	KP
KOREA, REPUBLIC OF	KR
KUWAIT	KW
KYRGYZSTAN	KG
LAO PEOPLE'S DEMOCRATIC REPUBLIC	LA
LATVIA	LV
LEBANON	LB
LESOTHO	LS
LIBERIA	LR

Country	Code
LIBYAN ARAB JAMAHIRIYA	LY
LIECHTENSTEIN	LI
LITHUANIA	LT
LUXEMBOURG	LU
MACAU	MO
MACEDONIA, THE FORMER YUGOSLAV REPUBLIC OF	MK
MADAGASCAR	MG
MALAWI	MW
MALAYSIA	MY
MALDIVES	MV
MALI	ML
MALTA	MT
MARSHALL ISLANDS	MH
MARTINIQUE	MQ
MAURITANIA	MR
MAURITIUS	MU
MAYOTTE	YT
MEXICO	MX
MICRONESIA	FM
MOLDOVA, REPUBLIC OF	MD
MONACO	MC
MONGOLIA	MN
MONTSERRAT	MS
MOROCCO	MA
MOZAMBIQUE	MZ
MYANMAR	MM
NAMIBIA	NA
NAURU	NR
NEPAL	NP
NETHERLANDS	NL
NETHERLANDS ANTILLES	AN
NEW CALEDONIA	NC
NEW ZEALAND	NZ
NICARAGUA	NI

Country	Code
NIGER	NE
NIGERIA	NG
NIUE	NU
NORFOLK ISLAND	NF
NORTHERN MARIANA ISLANDS	MP
NORWAY	NO
OMAN	OM
PAKISTAN	PK
PALAU	PW
PANAMA	PA
PAPUA NEW GUINEA	PG
PARAGUAY	PY
PERU	PE
PHILIPPINES	PH
PITCAIRN	PN
POLAND	PL
PORTUGAL	PT
PUERTO RICO	PR
QATAR	QA
REUNION	RE
ROMANIA	RO
RUSSIAN FEDERATION	RU
RWANDA	RW
SAINT KITTS AND NEVIS	KN
SAINT LUCIA	LC
SAINT VINCENT AND THE GRENADINES	VC
SAMOA	WS
SAN MARINO	SM
SAO TOME AND PRINCIPE	ST
SAUDI ARABIA	SA
SENEGAL	SN
SEYCHELLES	SC
SIERRA LEONE	SL
SINGAPORE	SG

Country	Code
SLOVAKIA (Slovak Republic)	SK
SLOVENIA	SI
SOLOMON ISLANDS	SB
SOMALIA	SO
SOUTH AFRICA	ZA
SPAIN	ES
SRI LANKA	LK
ST. HELENA	SH
ST. PIERRE AND MIQUELON	PM
SUDAN	SD
SURINAME	SR
SVALBARD AND JAN MAYEN ISLANDS	SJ
SWAZILAND	SZ
SWEDEN	SE
SWITZERLAND	CH
SYRIAN ARAB REPUBLIC	SY
TAIWAN, PROVINCE OF CHINA	TW
TAJIKISTAN	TJ
TANZANIA, UNITED REPUBLIC OF	TZ
THAILAND	TH
TOGO	TG
TOKELAU	TK
TONGA	TO
TRINIDAD AND TOBAGO	TT
TUNISIA	TN
TURKEY	TR
TURKMENISTAN	TM
TURKS AND CAICOS ISLANDS	TC
TUVALU	TV
UGANDA	UG
UKRAINIAN	UA
UNITED ARAB EMIRATES	AE
UNITED KINGDOM	GB
UNITED STATES	US

Country	Code
UNITED STATES MINOR OUTLYING ISLANDS	UM
URUGUAY	UY
UZBEKISTAN	UZ
VANUATU	VU
VATICAN CITY STATE (HOLY SEE)	VA
VENEZUELA	VE
VIET NAM	VN
VIRGIN ISLANDS (BRITISH)	VG
VIRGIN ISLANDS (U.S.)	VI
WALLIS AND FUTUNA ISLANDS	WF
WESTERN SAHARA	EH
YEMEN	YE
YUGOSLAVIA	YU
ZAIRE	ZR
ZAMBIA	ZM
ZIMBABWE	ZW

It should be noted that each locale also has a three-letter code and a numerical code defined by IS03166, but as OpenZinc Application Framework uses only the two-letter code we only present those here.

APPENDIX H - ISO LANGUAGE CODES

This appendix lists the ISO language codes. OpenZinc will maintain compatibility with the ISO definitions as they are updated or, in certain cases, before they are officially adopted if it is evident that a proposed standard will be adopted. Please be aware that the inclusion of a language code in this table does not imply support for that language code by OpenZinc Application Framework. This table is the complete ISO table.

Language Codes

These codes are used by OpenZinc for identifying a particular language. The language identified by these codes will be used when displaying text on objects in the library. The codes are from the IS0639 standard.

Language	Code
(AFAN) OROMO	om
ABKHAZIAN	ab
AFAR	aa
AFRIKAANS	af
ALBANIAN	sq
AMHARIC	am
ARABIC	ar
ARMENIAN	hy
ASSAMESE	as
AYMARA	ay
AZERBAIJANI	az
BASHKIR	ba
BASQUE	eu
BENGALI; BANGLA	bn
BHUTANI	dz
BIHARI	bh
BISLAMA	bi
BRETON	br
BULGARIAN	bg

Language	Code
BURMESE	**my**
BYELORUSSIAN	**be**
CAMBODIAN	**km**
CATALAN	**ca**
CHINESE	**zh**
CORSICAN	**co**
CROATIAN	**hr**
CZECH	**cs**
DANISH	**da**
DUTCH	**nl**
ENGLISH	**en**
ESPERANTO	**eo**
ESTONIAN	**et**
FAEROESE	**fo**
FIJI	**fj**
FINNISH	fi
FRENCH	**fr**
FRISIAN	fy
GALICIAN	**gl**
GEORGIAN	**ka**
GERMAN	**de**
GREEK	**el**
GREENLANDIC	**kl**
GUARANI	**gn**
GUJARATI	**gu**
HAUSA	**ha**
HEBREW	**iw**
HINDI	**hi**
HUNGARIAN	**hu**
ICELANDIC	**is**
INDONESIAN	**in**
INTERLINGUA	**ia**
INTERLINGUE	**ie**
INUPIAK	**ik**

Language	Code
IRISH	ga
ITALIAN	it
JAPANESE	ja
JAVANESE	jw
KANNADA	kn
KASHMIRI	ks
KAZAKH	kk
KINYARWANDA	rw
KIRGHIZ	ky
KIRUNDI	rn
KOREAN	ko
KURDISH	ku
LAOTHIAN	lo
LATIN	la
LATVIAN, LETTISH	lv
LINGALA	ln
LITHUANIAN	lt
MACEDONIAN	mk
MALAGASY	mg
MALAY	ms
MALAYALAM	ml
MALTESE	mt
MAORI	mi
MARATHI	mr
MOLDAVIAN	mo
MONGOLIAN	mn
NAURU	na
NEPALI	ne
NORWEGIAN	no
OCCITAN	oc
ORIYA	or
PASHTO, PUSHTO	ps
PERSIAN	fa
POLISH	pl

Language	Code
PORTUGUESE	**pt**
PUNJABI	**pa**
QUECHUA	**qu**
RHAETO-ROMANCE	**rm**
ROMANIAN	**ro**
RUSSIAN	**ru**
SAMOAN	**sm**
SANGRO	**sg**
SANSKRIT	**sa**
SCOTS GAELIC	**gd**
SERBIAN	**sr**
SERBO-CROATIAN	**sh**
SESOTHO	**St**
SETSWANA	**tn**
SHONA	**sn**
SINDHI	**sd**
SINGHALESE	**si**
SISWATI	**ss**
SLOVAK	**sk**
SLOVENIAN	**si**
SOMALI	**so**
SPANISH	**es**
SUNDANESE	**su**
SWAHILI	**sw**
SWEDISH	**sv**
TAGALOG	**tl**
TAJIK	**tg**
TAMIL	**ta**
TATAR	**tt**
TEGULU	**te**
THAI	**th**
TIBETAN	**bo**
TIGRINYA	**ti**
TONGA	**to**

Language	Code
TSONGA	ts
TURKISH	tr
TURKMEN	tk
TWI	tw
UKRAINIAN	uk
URDU	ur
UZBEK	uz
VIETNAMESE	vi
VOLAPUK	vo
WELSH	cy
WOLOF	wo
XHOSA	xh
YIDDISH	ji
YORUBA	yo
ZULU	zu

APPENDIX | - HARDWARE ISSUES

OpenZinc Application Framework provides low-level support for the following hardware configurations:

IBM AT and DOS/V
NEC PC 9800

Each of these systems operates differently at the hardware level. Because OpenZinc provides low-level hardware support for each environment, your applications will be readily portable.

NOTE: Because Japanese hardware is not IBM AT compatible, you should not make any BIOS calls. It will almost certainly not do what was intended.

Binding Device Drivers

The makefiles and library are configured, by default, to build libraries compatible with IBM AT-type hardware, including DOS/V. If support for the NEC PC 9800 is desired, or if an executable that may be used on either configuration (i.e., if run-time device driver binding is required) is needed, then you will need to rebuild the library for these purposes.

To build the library for the NEC PC 9800 hardware configuration only, you will need to define the **ZIL_HARDWARE** macro in the **UI_ENV.HPP** source module. You will also need to modify the makefile in the \OpenZinc\SOURCE directory. At the top of the makefile there are definitions for **DOSHARDWAREDEP** and **DOSHARDWARELIB.** For example, for the Microsoft compiler these are defined to be **i_mscat.obj** and **+i_mscat.obj,** respectively, or for the Borland compiler they are defined to be **i_btcat.obj** and **+i_btcat.obj.** If support for other compilers is implemented, they will use a similar naming convention. Change these to **i_mscnec.obj** and **+i_mscnec.obj** or **i_btcnec.obj** and **+i_btcnec.obj** as appropriate for the compiler you are using. Then use this makefile to build the libraries. Any application built using this library will be ready to run on an NEC PC 9800 hardware configuration.

To build the library so that it can bind the appropriate drivers at run-time for either the IBM-AT, including DOS/V, or the NEC PC 9800, the procedure is very similar to that described above. First define **ZIL_HARDWARE** in **UI_ENV.HPP.** But then, instead of changing the definitions for **DOSHARDWAREDEP** and **DOSHARDWARELIB,** add **i_mscnec.obj** or ijbtcnec.obj so that both the "AT" and "NEC" modules are present. Then rebuild the libraries. Any applications built using this library can be run on either

the IBM-AT or NEC PC 9800 hardware configurations. OpenZinc Application Framework will bind the appropriate drivers at run-time.

Macros

The following macros are used to initialize hardware support function names in the library. Their descriptions are provided for anyone needing to port the library to an unsupported environment. The programmer typically will never need to use these macros.

I_MAKENAME—Constructs a hardware-specific function name from a generic function name. The generic function name is passed to the macro as a parameter. In addition to the function name passed in, another macro, **ZIL_MODULE,** is used when constructing the hardware-specific name. **ZIL_MODULE** contains a value that identifies the hardware being used (see description of **ZIL_MODULE** below).

The hardware-specific function name is constructed by prepending the value of **ZIL_MODULE** on the generic function name. For example, if **ZIL_MODULE** is defined to be "NEC" (specifying an NEC 9800 hardware configuration), and the base function name is **I_ScreenOpen,** the resulting function name will be "NEC" + **I_ScreenOpen,** or **NECI_ScreenOpen.** This function is defined in the appropriate **I_*.CPP** file. For example, if the Borland compiler and NEC hardware are being used, the function would appear in the **I_BTCNEC.CPP** source code module.

The programmer should never need to use this macro. It is used by the library only.

MAKE_SETFUNCTIONS—Creates a function which will be used to convert all generic device driver function names to hardware-specific names. The name of the function is created by calling **I_MAKENAME** (see description above for details), passing it **I_SetFunctions** (the generic function name). Thus, if an NEC hardware configuration is being used, the function name generated will be **NECI_SetFunctions.**

The function defined by **MAKE_SETFUNCTIONS** simply calls **I_MAKENAME** for each generic driver function. The function is called at program initialization from within **UI_DISPLAY::I18NInitialize(**) once the hardware configuration is determined.

The programmer should never need to use this macro. It is used by the library only.

ZIL_HARDWARE—Indicates whether generic function names need to be converted to hardware-specific function names. If **ZIL_HARDWARE** is defined, conversion

will take place. If it is not defined, no conversion will occur. The library assumes IBM-AT hardware as the default configuration, so conversion is necessary only for non-AT hardware.

This macro must be defined in **UI_ENV.HPP** if building the library for non-AT hardware configurations.

ZIL_MODULE—Defines the hardware type in use. For example, if an NEC configuration is being used, **ZIL_MODULE** will be "NEC." This macro is used by the **I_MAKENAME** macro when constructing a hardware-specific function name from the generic function name (see example in **I_MAKENAME** macro description).

Unless creating device drivers for a new hardware environment, this macro should not be used by the programmer.

Generic Keyboard Functions

This section describes functions used for low-level interaction with the keyboard. These functions should not be used directly by the programmer, but descriptions are provided in the event that the programmer needs to implement a similar device, has need of implementing such a device on a new hardware configuration, or just wants to gain a better understanding of the operation of OpenZinc Application Framework.

I_KeyboardClose

Syntax

```
void I_KeyboardClose(void);
```

Remarks

This function restores the ISR that was in use prior to the initialization of the keyboard device and that was saved by **I_KeyboardOpen.** This function is called by the UID_KEYBOARD destructor.

I_KeyboardOpen

Syntax

void I_KeyboardOpen(void);

Remarks

This function initializes the keyboard device for input. It checks which type of keyboard is in use (e.g., enhanced keyboard or not) and saves the type for use by other functions. It also saves the <CTRL-C> and <CTRL-BREAK> interrupt service routine (ISR) currently in use and replaces it with an ISR from OpenZinc Application Framework's library. This function is called by the UID_KEYBOARD constructor.

I_KeyboardQuery

Syntax

void I_KeyboardQuery(unsigned **shiftState);*

Remarks

This function checks to see if there is a character in the keyboard buffer. If there is no keypress waiting, the current keyboard shift-state is returned.

- *returnValue*$_{out}$ indicates the state of the keyboard. If a keypress is waiting to be processed, *returnValue* is TRUE. If there are no keypresses waiting, it is FALSE.

- *shiftState*$_{out}$ is set to the IBM-defined shift state value (e.g., <CTRL> keypress is 0x0004). OpenZinc provides definitions for these values in the UI_EVT.HPP source module. For example, S_CTRL = 0x0004.

I_Keyboard Read

Syntax

void I_KeyboardRead(unsigned **rawCode,* unsigned **shiftState,* unsigned **value);*

Remarks

This function reads a character from the keyboard. If a character is not available, the function waits until a key is pressed. It is called by the UID_KEYBOARD::Poll() function.

- *rawCode*$_{in/out}$ is set to the raw, device-dependent scan-code of the keypress.

- *shiftState*$_{in/out}$ is set to the IBM-defined shift state value (e.g., <CTRL> keypress is 0x0004). OpenZinc provides definitions for these values in the UI_EVT.HPP source module. For example, S_CTRL = 0x0004.

- *value*$_{in/out}$ is set to the ASCII, ISO or Unicode character value, depending on the mode in which the program is running.

Generic Mouse Functions

This section describes functions used for low-level interaction with the mouse. These functions should not be used directly by the programmer, but descriptions are provided in the event that the programmer needs to implement a similar device, has need of implementing such a device on a new hardware configuration, or just wants to gain a better understanding of the operation of OpenZinc Application Framework.

I_MouseClose

Syntax

void I_MouseClose(void);

Remarks

This function de-initializes the mouse device and restores the mouse ISR saved by I_MouseOpen, if one existed. This function is called by the UID_MOUSE destructor.

I_MouseOpen

Syntax

int I_MouseOpen(void);

Remarks

This function initializes the mouse device. This function saves the current mouse ISR, if one exists, and sets the ISR to one provided by OpenZinc Application Framework's library (MouseISR( )). This function is called by the UID_MOUSE constructor.

- *returnValue*$_{out}$ indicates the status of the mouse device after attempting to initialize it. If the mouse device was successfully initialized, *returnValue* is TRUE. Otherwise it is FALSE.

In this chapter we discuss the specifics of the text mode device driver. This includes the functions to control the edit cursor in text mode.

Global Variables

The following global variables define text characters used to draw objects in the library:

ZIL_ICHAR _tLowerShadow[]—Defines the lower-box character used as the shadow in the upper-right corner of a window. The default definition in IBM English mode is "▄".

ZIL_ICHAR _tUpperShadow[]—Defines the upper-box character used as the shadow along the lower edge of a window. The default definition in IBM English mode is "▀".

ZIL_ICHAR _tFullShadow[]—Defines the full-box character used as the shadow along the right edge of a window. The default definition in IBM English mode is "█".

Text Driver Functions

This section describes functions used for low-level interaction with the text display. These functions should not be used directly by the programmer but descriptions are provided in the event that the programmer needs to implement a similar device, has need of implementing such a device on a new hardware configuration, or just wants to gain a better understanding of the operation of OpenZinc Application Framework.

I_ScreenClose

Syntax

void I_ScreenClose(void);

Remarks

This function restores the screen to the mode it was in prior to running the application. The original mode was saved by I_ScreenOpen(). This function also restores the blink attribute. This function is called by the UI_TEXT_DISPLAY destructor.

I_ScreenOpen

Syntax

void I_ScreenOpen(int **mode*, int **lines*, int **columns);*

Remarks

This function initializes the screen device for use. It saves the mode in use at startup and turns off the blink attribute to allow the use of 16 colors. This function is called by the UI_TEXT_DISPLAY constructor.

- $mode_{in/out}$ specifies what resolution to use. When returning from this function, *mode* will contain the mode actually initialized, *mode* can be any valid text display mode value as defined by IBM. OpenZinc defines constants for each value (e.g., TDM_25x40 = 1) in the UI_DSP.HPP source file. For a complete listing of TDM_ values, see "Chapter 40—UI_TEXT_DISPLAY" in *Programmer's Reference Volume 1* or see the *Quick Reference Guide.*

- $lines_{out}$ will contain the number of horizontal lines on the initialized display.

- $columns_{out}$ will contain the number of vertical columns on the initialized display.

I_ScreenPut

Syntax

void I_ScreenPut(int *left*, int *top*, int *right*, int *bottom*, void **buffer);*

Remarks

This function places the contents of a buffer on the screen. This function is called from the UI_TEXT_DISPLAY constructor, **UI_TEXT_DISPLAY::DeviceMove()**, **UI_-TEXT_DISPLAY::RegionMove()**, **UI_TEXT_DISPLAY::VirtualGet()**, and **UI_-TEXT_DISPLAY::VirtualPut().**

- $left_{in}$, top_{in}, $right_{in}$, $bottom_{in}$ is the region of the screen where the buffer is to be displayed, *left* and *top* are zero-based, so the upper-left corner of the screen is at 0, 0.

- $buffer_{in}$ is really a pointer to short values. The high-order byte of each short specifies the foreground and background colors to display the character (i.e., the high-order nibble sets the background and the low-order nibble sets the foreground), and the low-order byte contains the character to be displayed.

I_CursorPosition

Syntax

void I_CursorPosition(int *y,* int *x,* int *val);*

Remarks

This function positions the edit cursor, or caret, on the screen.

NOTE: The edit cursor should not be confused with the mouse cursor.

- y_{in} is the y, or vertical, position of the cursor on the screen. The screen is zero-based, so the top row is 0.

- x_{in} is the x, or horizontal, position of the cursor on the screen. The screen is zero-based, so the left column is 0.

- val_m indicates the type of cursor to display. If *val* is DC_INSERT, the cursor is an insert cursor. Otherwise the cursor is an overstrike cursor.

I_CursorRemove

Syntax

void I_CursorRemove(void);

Remarks

This function removes the edit cursor, or caret, from the screen. To redisplay the cursor, call I_CursorPosition.

NOTE: The edit cursor should not be confused with the mouse cursor.

Generic Internationalization Functions

This section describes a function used for low-level interaction with the operating system. This function should not be used directly by the programmer.

I_GetCodePage

Syntax

void I_GetCodePage(void);

Remarks

This function determines which code page is in use by the operating system.

INDEX

<CTRL-BREAK>
ISR 806
<CTRL-C>
ISR 806
_applicationIconName 208
_asteriskIconName 209
_delta 441-443
_height 459,460
_numberID 681, 704
_userFunction 441, 443
_value 781
_woFlags 459, 460

A

Add (function) 91, 188, 281, 303, 324, 363, 382, 660, 688
application
for NEC PC 9800 803
application icon 208
applications
run-time driver binding 804
arg 746,750,751
argl 751,752
arg2 751,752
arrays
of pop-up items 323, 361, 381
pop-up menu use of 323
pull-down item use of 361
pull-down menu use of 381
asterisk icon 209
atcFlags 774
attrib (macro) 747

B

background 161, 747, 769, 810
bdFlags 39,40
bignum 15
BIOS calls 803
bitmapArray 56, 58, 61
bitmapHeight 56, 58
bitmapName 55-58, 63, 776
bitmapWidth 56, 57
border 39
borderWidth 279,280
btFlags 55, 57, 58, 295, 297, 776
btFlags (variable) 57
buffer
keyboard 806
button 55
getting text 74
maximize 251
minimize 265
seting text 75
system 499

C

caret 810
cellHeight 181-183, 186, 777
character
checking the keyboard 806
getting from keyboard 806
Check boxes
BTF_CHECK_BOX (flag) 59
CheckSelection (function) 690
child windows 680
ClassName
pop-up item implementation of 304
ClassName (function)
bignum implementation of 23
button implementation of 65
ClassName (virtual function)
border implementation of 42
combo box implementation of 92
date implementation of 121

formatted string implementation of 144
group implementation of 166
horizontal list implementation of 189
icon implementation of 214
integer implementation of 238
maximize button implementation of 254
minimize button implementation of 268
notebook implementation of 282
pop-up menu implementation of 325
prompt implementation of 342
pull-down item implementation of 364
pull-down menu implementation of 383
real implementation of 402
scroll bar implementation of 426
spin control implementation of 446
status bar implementation of 462
string implementation of 479
system button implementation of 504
text implementation of 577
time implementation of 605
title implementation of 625
tool bar implementation of 642
vertical list implementation of 661
window implementation of 690
clipList 681, 683
colorBitmap 56, 58
columnHeader 520, 521, 532
combo box 85
getting text 102
setting text 102
comboShell 672
compare function
auto sort 712
compare functions
combo box 90
compareFunction 85, 87, 88, 90, 104, 181, 183, 184, 186, 198, 653, 655, 656, 658, 670
compressedText 137-140, 144, 145, 776
construction order of objects 253, 267, 503
Control (function) 725, 735
Count (function) 93
Current (function) 94, 691
currentRecord 520, 521
cursor
I_CursorPosition 810
I_CursorRemove 811
position 810
removing from the screen 811
CursorOffset (function) 578

D

DataGet (function) 23, 43, 66, 122, 144, 166, 215, 238, 343, 402, 480, 524, 578, 605, 625
bignum implementation of 23
border implementation of 43
button implementation of 66
date implementation of 122
formatted string implementation of 144
group implementation of 166
icon implementation of 215
integer implementation of 238
prompt implementation of 343
real implementation of 402
string implementation of 480
text implementation of 578
time implementation of 605
title implementation of 625
DataSet (function) 25, 43, 67, 123, 146, 167, 216, 239, 344, 403, 481, 525, 579, 606, 626
bignum implementation of 25
border implementation of 43
button implementation of 67
date implementation of 123
formatted string implementation of 146
group implementation of 167
icon implementation of 216
integer implementation of 239
prompt implementation of 344
real implementation of 403, 405
string implementation of 481
text implementation of 579
time implementation of 606
title implementation of 626
date 113
day 116-118,129,745

decimal 15, 18, 22, 395, 398, 406, 785, 786, 787
decimal values (fixed place) 18
decorationName 56, 77, 252, 258, 266, 271, 296, 310, 500, 510
defaultInitialized 15, 16, 55-57, 113, 114, 231, 232, 251, 252, 265, 266, 295, 296, 395, 396, 441, 442, 500, 501, 597, 598, 680, 682
defaultObject 681,683
defFlag 733-735
DeleteRecord (function) 526
deleteText 137-140, 145, 146, 148, 149, 151, 776
denominator 775
depth 56, 57, 286, 309, 369, 509, 704, 707, 776
destination 137, 149, 150
Destroy (function) 94, 190, 662, 691
device driver
  binding 803
  cursor 810
  hardware 803
  initializing 804
  keyboard 805
  mouse 807
  text display 809
dncFlags 774
DOS/V
  devices 803
DOSHARDWAREDEP 803
DOSHARDWARELIB 803
DrawItem (function)
  border implementation of 44
  button implementation of 68
  icon implementation of 216
  pop-up item implementation of 304
  prompt implementation of 345
  pull-down item implementation of 365
  string implementation of 482
  text implementation of 580
DrawItem (virtual function)
  notebook implementation of 283
  scroll bar implementation of 427
  status bar implementation of 463
  table header implementation of 544
  table implementation of 526
  table record implementation of 559
  window implementation of 692
DrawRecord (function) 527
dtFlags 113-115,491,776
dtFlags (variable) 114

E

edit fields
  combo box 85
  date 113
  floating point 395
  formatted strings 137
  integer 231
  multi-line text 571
  numeric 17
  single line text 473
  time 597
edit mask 139
editMask 137-139, 145, 146, 148, 149, 151, 776
element1 88, 184, 656
element2 88, 184, 656
Event (virtual function)
  bignum implementation of 26
  border implementation of 45
  button implementation of 69
  combo box implementation of 95
  date implementation of 124
  floating-point implementation of 404
  formatted string implementation of 147
  group implementation of 168
  horizontal list implementation of 190
  icon implementation of 217
  integer implementation of 240
  maximize button implementation of 255
  minimize button implementation of 269
  notebook implementation of 284
  pop-up item implementation of 305
  pop-up menu implementation of 326
  prompt implementation of 346
  pull-down item implementation of 366
  pull-down menu implementation of 383

real number implementation of 404
scroll bar implementation of 428
spin control implementation of 446
status bar implementation of 464
string implementation of 483
system button implementation of 505
table header implementation of 545
table implementation of 528
table record implementation of 560
text implementation of 581
time implementation of 607
title implementation of 627
tool bar implementation of 643
vertical list implementation of 662
window implementation of 693
events
logical 763
system 757
exclamation icon 209
expanded 137, 149, 150
Export (function) 149
formatted string implementation of 149

F

fieldObject 441-443,450,451,779
file
storage 773
First (function) 98, 699
flagl 748, 749
flag2 748, 749
flags
button 59
date 116
formatted strings 140
icon 210
integer 234
numeric 18
pop-up item 298, 301
pop-up menu 321, 323
scroll bar 420, 422
single line text 476
slider 422
time 600
FlagSet (macro) 748
flagSetting 85, 87, 90, 181, 183, 186, 653, 655, 658
FlagsSet (macro) 749
floating-point values 395
foreground 747, 810
Format (function) 405
formatted string 137

G

Generic (function) 507, 700
system button implementation of 507
window implementation of 700
Get (function) 99, 704
GetCursorPos (function) 587
GetRecord (function) 530
group
getting text 171
setting text 172
group box 161

H

hand icon 209
hardware
DOS/V 803
IBM AT 803
Japanese 803
NEC PC 9800 803
supported 803
help contexts 686, 703, 725
helpContext 507, 680, 681, 683, 686, 700, 703, 721, 722, 725, 775
horizontal list 181
getting text 195
hScroll 193, 197, 419, 430, 487, 529, 586, 681, 683, 697, 708, 711, 758, 759
hScrollInfo 681,683,711

I

I_CursorPosition
  function 810
I_CursorRemove
  function 811
LKeyboardClose
  function 805
I_KeyboardOpen
  function 806
I_KeyboardQuery
  function 806
I_KeyboardRead
  function 806
I_MAKENAME
  macro 804
  use of 804
I_MouseClose
  function 807
I_MouseOpen
  function 808
I_ScreenClose
  function 809
I_ScreenOpen
  function 809
I_ScreenPut
  function 810
IBM AT
  devices 803
icFlags (variable) 207, 209, 210, 777, 209
icon 207
  default names 208
  getting text 222
  setting text 222
iconArray 208,209,211,777
iconHeight 208, 209, 777
iconName 207-210, 777
iconRegion 208, 209
iconWidth 208, 209, 777
identifiers 769
Import (function) 151
  formatted string implementation of 151
include file
  UI_DSP.HPP 7
  UI_EVT.HPP 7
  UI_GEN.HPP 6
  ULMAP.HPP 7
  UI_WIN.HPP 8
indentation 377-379
index 86, 99, 100, 813
Index (function) 99
Information (virtual function)
  bignum implementation of 27
  border implementation of 47
  button implementation of 72
  combo box implementation of 100
  date implementation of 125
  formatted string implementation of 152
  group implementation of 170
  horizontal list implementation of 194
  icon implementation of 220
  integer implementation of 241
  maximize button implementation of 256
  minimize button implementation of 270
  notebook implementation of 286
  pop-up item implementation of 308
  pop-up menu implementation of 328
  prompt implementation of 347
  pull-down item implementation of 368
  pull-down menu implementation of 385
  real implementation of 406
  scroll bar implementation of 431
  spin control implementation of 449
  status bar implementation of 465
  string implementation of 488
  system button implementation of 508
  table header implementation of 547
  table implementation of 530
  table record implementation of 561
  text implementation of 588
  time implementation of 608
  title implementation of 629
  tool bar implementation of 645
  vertical list implementation of 666
  window implementation of 705
insertMode 473,474,571,572
InsertRecord (function) 533
integer 231
ISR
  <CTRL-BREAK> 806
  <CTRL-C> 806
  keyboard 805

mouse 807, 808
ItemDepthSearch (function) 386

J

Japanese hardware 803

K

keyboard
checking 806
closing 805
device driver 805
getting character 806
I_KeyboardClose 805
I_KeyboardOpen 806
I_KeyboardQuery 806
I_KeyboardRead 806
initializing 806
ISR 805
shift state 806

L

labelWidget 161, 162
languageName 16, 29, 114, 127, 232, 243, 396, 408, 442, 500, 511, 598, 611, 681, 684, 687, 711
Last (function) 103,709
libraries
NEC PC 9800 803
list 181, 653
list (horizontal) 181
list (vertical) 653
Load (virtual function)
bignum implementation of 33
border implementation of 49
button implementation of 79
combo box implementation of 106
date implementation of 131
dialog window implementation of 728
formatted string implementation of 156
group implementation of 175
horizontal list implementation of 201
icon implementation of 224
integer implementation of 246
maximize button implementation of 260
minimize button implementation of 274
notebook implementation of 289
pop-up item implementation of 313
pop-up menu implementation of 332
prompt implementation of 350
pull-down item implementation of 372
pull-down menu implementation of 389
real implementation of 411
scroll bar implementation of 435
spin control implementation of 453
status bar implementation of 468
string implementation of 493
system button implementation of 513
table header implementation of 550
table implementation of 536
table record implementation of 565
text implementation of 592
time implementation of 615
title implementation of 632
tool bar implementation of 648
vertical list implementation of 673
window implementation of 716

M

macros 804
MAKE_SETFUNCTIONS
macro 804
maximize button 251
maximizing a window 251
maxRecords 519-523, 525, 541, 542
maxValue 3,24,31,474,491,751-753
MDI windows 680, 685, 702
menu items
pop-up 295

pull-down 357
menus
pop-up 319
pull-down 377
tool bar 637
Message (function) 76
button implementation of 76
minimize button 265
minimizing a window 265
miniNumeratorX 782
minObject 507, 680, 684, 700
minValue 3, 474, 491, 752, 753
mniFlags 295-297, 778
mniFlags (variable) 298, 301
monoBitmap 56, 58
month 116-118, 121, 129, 604, 745
mouse
closing 807
device driver 807
I_MouseClose 807
I_MouseOpen 808
initializing 808
ISR 807, 808
moving a window 621
mSelect 378, 386, 387
msgFlags 733-735
multi-line text 571
multiple inheritance
window implementation of 679
myLanguage 16, 17, 30, 114, 115, 128, 232, 243, 396, 408, 500, 501, 511, 598, 599, 611, 681, 683, 712

N

NEC PC 9800
devices 803
New (function)
bignum implementation of 34
border implementation of 50
button implementation of 80
combo box implementation of 108
date implementation of 133
dialog window implementation of 729
formatted string implementation of 157
group implementation of 176
horizontal list implementation of 202
icon implementation of 226
integer implementation of 247
maximize button implementation of 261
minimize button implementation of 275
notebook implementation of 290
pop-up item implementation of 315
pop-up menu implementation of 333
prompt implementation of 352
pull-down item implementation of 373
pull-down menu implementation of 391
real implementation of 412
scroll bar implementation of 436
spin control implementation of 454
status bar implementation of 469
string implementation of 495
system button implementation of 514
table header implementation of 551
table implementation of 537
table record implementation of 566
text implementation of 593
time implementation of 616
title implementation of 633
tool bar implementation of 649
vertical list implementation of 675
window implementation of 717
NewFunction (virtual function)
bignum implementation of 36
border implementation of 51
button implementation of 82
combo box implementation of 109
formatted string implementation of 158
group implementation of 177
horizontal list implementation of 203
icon implementation of 227
integer implementation of 249
maximize button implementation of 262
minimize button implementation of 276
notebook implementation of 291
pop-up item implementation of 316
pop-up menu implementation of 334
prompt implementation of 353
pull-down item implementation of 375
pull-down menu implementation of 392

real implementation of 413
scroll bar implementation of 437
spin control implementation of 455
status bar implementation of 471
string implementation of 496
system button implementation of 516
table header implementation of 552
table implementation of 538
table record implementation of 567
text implementation of 595
time implementation of 617
title implementation of 635
tool bar implementation of 650
vertical list implementation of 676
window implementation of 719
nmFlags 15, 17, 231-233, 395-397, 775, 777, 779
nmFlags (variable) 18
noOfObjects 781, 782
number 15
integer representation of 231
numberlD 8, 101, 286, 287, 309, 369, 509, 681, 704, 707, 742, 774, 775
numerator 775

O

objectl 681,712-714
object2 681,712-714
offset 572, 578, 583-585, 588, 590, 765, 774, 775
operator overload
+ 91, 281, 303, 363, 688
- 104, 198, 311, 329, 370, 387, 670, 713
output
text display 810

P

ParseRange (function) 491
pixmap 56, 58
pop-up item 295
pop-up menu 319
processError 16, 30, 31, 114, 128, 129, 231, 244, 245, 395, 408, 409, 491, 598, 611, 612
prompt 337
getting text 348
setting text 348
pull-down item 357
pull-down menu 377

Q

question icon 209
Quick 809

R

radio buttons
BTF_RADIO_BUTTON (flag) 60
range 15-19, 21, 24, 28, 30, 31, 113-116, 118, 120, 126, 129, 141, 231, 232-234, 236, 242, 244, 395-399, 401, 407, 409, 417, 421, 441, 443, 444, 491, 492, 597-602, 604, 612, 726, 775-777, 779, 781
ranges
date 116
integer 233
numeric 18
time use of 600
rawCode 541, 555, 558, 561, 760, 763, 806
recordNum 520, 526, 527, 530, 533, 534, 556
records 519-526,532,534,543,555
recordSize 519-522, 541, 542
refObjectID 774
RegionMax (function) 172, 562, 668, 709
rlcFlags 774

rowHeader 520, 521, 533
run-time driver binding 803

S

SampleFunction (function) 2
sbFlags 9, 417-419, 779
sbFlags (variable) 420, 422
scroll bar
  horizontal 417
  vertical 417
ScrollEvent (function) 197, 669, 710
searchID 774, 778, 779, 782
selectable objects
  button 55
  icon 207
  maximize button 251
  minimize button 265
  pop-up item 295
  pull-down item 357
  system button 499
  title bar 621
selectedObject 681, 690, 705
SetCurrent (function) 534
SetCursorPos (function) 590
SetDecorations (function) 77, 258, 271, 310, 510
SetLanguage (function) 29, 127, 243, 408, 511, 611, 711
shadow border 808
shadowWidth 279,280
shift state
  checking the keyboard 806
shiftState 8, 806
signature 773
single-line text 473
sizing a window 39
Slider 417
Sort (function) 103, 198, 670
source 30, 77, 128, 137, 151, 243, 258, 272, 311, 408, 511, 611, 712, 739, 803, 804, 806, 807, 809
spin control
  getting text 450
  setting text 450
static variables
  width 40
status
  button 56, 57
stFlags 473-475, 780
stFlags (variable) 476
storage 773
Store (virtual function) 52, 134
  bignum implementation of 36
  border implementation of 52
  button implementation of 82
  combo box implementation of 110
  date implementation of 134
  dialog window implementation of 730
  formatted string implementation of 159
  group implementation of 178
  horizontal list implementation of 204
  icon implementation of 228
  integer implementation of 249
  maximize button implementation of 263
  minimize button implementation of 277
  notebook implementation of 292
  pop-up item implementation of 317
  pop-up menu implementation of 335
  prompt implementation of 354
  pull-down item implementation of 375
  pull-down menu implementation of 393
  real implementation of 414
  scroll bar implementation of 438
  spin control implementation of 456
  status bar implementation of 471
  string implementation of 497
  system button implementation of 516
  table header implementation of 553
  table implementation of 539
  table record implementation of 568
  text implementation of 595
  time implementation of 618
  title implementation of 635
  tool bar implementation of 651
  vertical list implementation of 677
  window implementation of 719
string 473
  getting text 489
  setting text 490

StringCompare (function) 712
  window implementation of 712
stringID 101, 286, 287, 309, 369, 509, 704, 707, 775
stripText 55, 66
Subtract (function) 104, 198, 311, 329, 370, 387, 670, 713
supportDecorations 681, 683
syFlags 500, 501, 780
system button 499

T

tableRecord 520, 521, 533
tblFlags 9, 519, 521, 522, 780
text 571
  getting text 589
  setting text 589
text display 809
  closing 809
  I_ScreenClose 809
  I_ScreenOpen 809
  I_ScreenPut 810
  initializing 809
  output 810
text mode
  blink attribute 809
  characters 808
thFlags 9, 541, 542, 780
time 597
title 621
  getting text 630
  setting text 630
titleRegion 208, 210
tmFlags 597-599,781
tool bar 637
topRecord 520, 522
Top Widget (function) 671

U

UI_TEXT_DISPLAY 809
UID_KEYBOARD 806
UIW_BIGNUM (class) 15
UIW_BIGNUM (function) 17, 32
UIW_BORDER (class) 39
UIW_BORDER (function) 40, 47
UIW_BUTTON (class) 55
UIW_BUTTON (function) 58, 78
UIW_COMB0_BOX (class) 85
UIW_COMB0_BOX (function) 87, 105
UIW_DATE (class) 113
UIW_DATE (function) 115,130
UIW_FORMATTED_STRING (class) 137
UIW_FORMATTED_STRING (function) 139, 154
UIW_GROUP (class) 161
UIW_GROUP (function) 162, 173
UIW_HZ_LIST (class) 181
UIW_HZ_LIST (function) 183, 199
UIW_ICON (class) 207
UIW_ICON (function) 223
UIW_INTEGER (class) 231
UIW_INTEGER (function) 233
UIW_MAXIMIZE_BUTTON (class) 251
UIW_MAXIMIZE_BUTTON (function) 253, 259
UIW_MINIMIZE_BUTTON (class) 265
UIW_MINIMIZE_BUTTON (function) 266, 272
UIW_NOTEBOOK (class) 279
UIW_NOTEBOOK (function) 280, 288
UIW_POP_UP_ITEM (class) 295
UIW_POP_UP_ITEM (function) 297, 312
UIW_POP_UP_MENU (class) 319
UIW_POP_UP_MENU (function) 320, 330
UIW_PROMPT (class) 337
UIW_PROMPT (function) 338, 349
UIW_PULL_DOWN_ITEM (class) 357
UIW_PULL_DOWN_ITEM (function) 358, 371
UIW_PULL_DOWN_MENU (class) 377
UIW_PULL_DOWN_MENU (function) 379, 388

UIW_REAL (class) 395
UIW_REAL (function) 397
UIW_SCROLL_BAR (class) 417
UIW_SCROLL_BAR (function) 419, 433
UIW_SPIN_CONTROL (class) 441
UIW_SPIN_CONTROL (function) 442, 451
UIW_STATUS_BAR (class) 459
UIW_STATUS_BAR (function) 460, 467
UIW_STRING (class) 473
UIW_STRING (function) 475, 492
UIW_SYSTEM_BUTTON (class) 499
UIW_SYSTEM_BUTTON (function) 501, 512
UIW_TABLE (class) 519
UIW_TABLE (function) 522, 535
UIW_TABLE_HEADER (class) 541
UIW_TABLE_HEADER (function) 542, 549
UIW_TABLE_RECORD (class) 555
UIW_TABLE_RECORD (function) 556, 564
UIW_TEXT (class) 571
UIW_TEXT (function) 573, 591
UIW_TIME (class) 597
UIW_TIME (function) 599, 613
UIW_TITLE (class) 621
UIW_TITLE (function) 622, 631
UIW_TOOL_BAR (class) 637
UIW_TOOL_BAR (function) 638, 646
UIW_VT_LIST (class) 653
UIW_VT_LIST (function) 655, 672
UIW_WINDOW (class) 679
UIW_WINDOW (function) 683, 715
user function
  bignum use of 20
  button use of 62
  date use of 119
  formatted string use of 142
  icon use of 212
  integer use of 235
  pop-item use of 301
  pop-up item use of 299
  pull-down item use of 360
  real use of 400
  scroll bar use of 423
  spin control use of 444
  string use of 477
  table record use of 557
  text use of 575
  time use of 603
user function definition 20, 62, 119, 142, 212, 235, 299, 360, 400, 423, 444, 477, 557, 575, 603
userFunction 15, 17, 21, 55, 58, 62, 113, 115, 120, 137, 139, 142, 207, 210, 212, 231, 233, 236, 295, 297, 300, 357, 358, 360, 395, 397, 400, 401, 418, 419, 423, 441, 443, 444, 473, 475, 477, 555-557, 571, 573, 575, 597, 599, 603, 604, 773

V

Validate (virtual function)
  bignum implementation of 30
  date implementation of 128
  integer implementation of 244
  real implementation of 408
  time implementation of 611
vertical list
  getting text 667
  setting text 667
virtualRecord 520, 521, 533, 556, 563
VirtualRecord (function) 563
vScroll 419, 430, 529, 587, 666, 669, 681, 683, 699, 708, 711, 761
vScrollInfo 197,669,681,683,711

W

width (static variable) 40
window 679
  child 680
  getting text 707
  MDI 680
  setting text 708

windows
text mode 808
wnFlags 85, 87, 103, 161, 162, 181, 183, 319, 320, 357, 358, 441, 443, 451, 571, 573, 637, 638, 653, 655, 680, 682, 779, 781, 782
woAdvancedFlags 85, 87, 181, 183, 319, 320, 377, 379, 418, 419, 423, 507, 637, 638, 653, 655, 680, 683, 700, 721, 722, 775
woAdvancedFlags (variable) 423
woStatus 680

Y

year 116-118,745

Z

ZAF_DIALOG_WINDOW (class) 721
ZAF_DIALOG_WINDOW (function) 722, 726
ZAF_MESSAGE_WINDOW (class) 733
ZAF_MESSAGE_WINDOW (function) 734
ZIL_HARDWARE 803
macro 804
ZIL_MODULE
macro 805
use of 804
ZIL_NULLH 3, 6, 753, 754
ZILJVOIDF 3, 6, 755
ZIL_VOIDP 3, 6, 755, 756
OpenZinc Application Framework
default icons 208

~UIW_BIGNUM (function) 22
~UIW_BUTTON (function) 64
~UIW_COMBO_BOX (function) 91
~UIW_DATE (function) 121
~UIW_FORMATTED_STRING (function) 143
~UIW_GROUP (function) 165
~UIW_HZ_LIST (function) 188
~UIW_ICON (function) 214
~UIW_INTEGER (function) 237
~UIW_MAXIMIZE_BUTTON (function) 254
~UIW_MINIMIZE_BUTTON (function) 267
~UIW_NOTEBOOK (function) 281
~UIW J O P J J P _ I T E M (function) 302
~UIW_POP_UP_MENU (function) 324
~UIW_PROMPT (function) 342
~UIW_PULL_DOWN_ITEM (function) 362
~UIW_PULL_DOWN_MENU (function) 381
~UIW_REAL (function) 401
~UIW_SCROLL_BAR (function) 425
~UIW_SPIN_CONTROL (function) 445
~UIW_STATUS_BAR (function) 462
~UIW_STRING (function) 479
~UIW_SYSTEM_BUTTON (function) 503
~UIW J A B L E (function) 524
~UIWJABLE_HEADER (function) 544
~UIW J E X T (function) 576
~UIW_TIME (function) 604
~UIW J I T L E (function) 624
~UIWJOOL_BAR (function) 641
~UIW_VT LIST (function) 660
~UIW_WINDOW (function) 688

GNU Free Documentation License
Version 1.3, 3 November 2008

Copyright (C) 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
<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. Secondarily, 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 such 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.