

OpenZinc

Programmers Reference  
Volume 1

Support Objects

Version 1

# Programmer's Reference

Volume One  
Support Objects

**OpenZinc® Application Framework**  
**Version 1.0**  
OpenZinc Software

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

<b>INTRODUCTION</b> .....	<b>1</b>
UI_SAMPLE_CLASS::SampleFunction	
<b>CLASSES AND STRUCTURES</b>	
<b>INCLUDE FILE HIERARCHY</b>	
<b>CLASS HIERARCHY</b>	
<b>CHAPTER 1 - UI_APPLICATION</b> .....	<b>15</b>
General Members	
UI_APPLICATION::UI_APPLICATION	
UI_APPLICATION::~~UI_APPLICATION	
UI_APPLICATION::Control	
UI_APPLICATION::LinkMain	
UI_APPLICATION::Main	
<b>CHAPTER 2 - UI_ATTACHMENT</b> .....	<b>25</b>
General Members	
UI_ATTACHMENT::UI_ATTACHMENT	
UI_ATTACHMENT::~~UI_ATTACHMENT	
UI_ATTACHMENT::Information	
UI_ATTACHMENT::Modify	
Storage Members	
UI_ATTACHMENT::UI_ATTACHMENT	
UI_ATTACHMENT::Load	
UI_ATTACHMENT::New	
UI_ATTACHMENT::NewFunction	
UI_ATTACHMENT::Store	
<b>CHAPTER 3 - UI_BGI_DISPLAY</b> .....	<b>37</b>
General Members	
UI_BGI_DISPLAY::UI_BGI_DISPLAY	
UI_BGI_DISPLAY::~~UI_BGI_DISPLAY	
UI_BGI_DISPLAY::SetFont	
UI_BGI_DISPLAY::SetPattern	
<b>CHAPTER 4 - UI_CONSTRAINT</b> .....	<b>45</b>
General Members	
UI_CONSTRAINT::UI_CONSTRAINT	
UI_CONSTRAINT::~~UI_CONSTRAINT	

UI\_CONSTRAINT::Information  
 UI\_CONSTRAINT::Manager  
 UI\_CONSTRAINT::Modify  
 UI\_CONSTRAINT::Next  
 UI\_CONSTRAINT::Previous  
 UI\_CONSTRAINT::SearchID  
 Storage Members  
 UI\_CONSTRAINT::UI\_CONSTRAINT  
 UI\_CONSTRAINT::Load  
 UI\_CONSTRAINT::New  
 UI\_CONSTRAINT::NewFunction  
 UI\_CONSTRAINT::Store

**CHAPTER 5 - UI\_DEVICE. . . . . 59**

General Members  
 UI\_DEVICE::UI\_DEVICE  
 UI\_DEVICE::~~UI\_DEVICE  
 UI\_DEVICE::CompareDevices  
 UI\_DEVICE::Event  
 UI\_DEVICE::Next  
 UI\_DEVICE::Poll  
 UI\_DEVICE::Previous

**CHAPTER 6 - UI\_DIMENSION\_CONSTRAINT. . . . . 71**

General Members  
 UI\_DIMENSION\_CONSTRAINT::UI\_DIMENSION\_CONSTRAINT  
 UI\_DIMENSION\_CONSTRAINT::~~UI\_DIMENSION\_CONSTRAINT  
 UI\_DIMENSION\_CONSTRAINT::Information  
 UI\_DIMENSION\_CONSTRAINT::Modify  
 Storage Members  
 UI\_DIMENSION\_CONSTRAINT::UI\_DIMENSION\_CONSTRAINT  
 UI\_DIMENSION\_CONSTRAINT::Load  
 UI\_DIMENSION\_CONSTRAINT::New  
 UI\_DIMENSION\_CONSTRAINT::NewFunction  
 UI\_DIMENSION\_CONSTRAINT::Store

**CHAPTER 7 - UI\_DISPLAY. . . . . 83**

General Members  
 UI\_DISPLAY::UI\_DISPLAY  
 UI\_DISPLAY::~~UI\_DISPLAY  
 UI\_DISPLAY::Bitmap  
 UI\_DISPLAY::BitmapArrayToHandle  
 UI\_DISPLAY::BitmapHandleToArray

UI\_DISPLAY::Ellipse  
 UI\_DISPLAY::IconArrayToHandle  
 UI\_DISPLAY::IconHandleToArray  
 UI\_DISPLAY::Line  
 UI\_DISPLAY::MapColor  
 UI\_DISPLAY::Polygon  
 UI\_DISPLAY::Rectangle  
 UI\_DISPLAY::RectangleXORDiff  
 UI\_DISPLAY::RegionDefine  
 UI\_DISPLAY::RegionInitialize  
 UI\_DISPLAY::RegionMove  
 UI\_DISPLAY::Text  
 UI\_DISPLAY::TextHeight  
 UI\_DISPLAY::TextWidth  
 UI\_DISPLAY::VirtualGet  
 UI\_DISPLAY::VirtualPut

**CHAPTER 8 - UI\_ELEMENT.....125**

General Members  
 UI\_ELEMENT::UI\_ELEMENT  
 UI\_ELEMENT::UI\_ELEMENT  
 UI\_ELEMENT::ClassName  
 UI\_ELEMENT::Information  
 UI\_ELEMENT::ListIndex  
 UI\_ELEMENT::Next  
 UI\_ELEMENT: Previous

**CHAPTER 9 - UI\_ERROR.STUB.....133**

General Members  
 UI\_ERROR.STUB::UI\_ERROR.STUB  
 UI\_ERROR.STUB::Beep  
 UI\_ERROR.STUB::ErrorMessage  
 UI\_ERROR.STUB::ReportError

**CHAPTER 10 - UI\_ERROR.SYSTEM.....139**

General Members  
 UI\_ERROR\_SYSTEM::UI\_ERROR\_SYSTEM  
 UI\_ERROR\_SYSTEM::~~UI\_ERROR\_SYSTEM  
 UI\_ERROR\_SYSTEM::ErrorMessage  
 UI\_ERROR\_SYSTEM::SetLanguage

**CHAPTER 11 - UI\_EVENT.....145**

General Members

UI_EVENT::UI_EVENT	
UI_EVENT::InputType	
<b>CHAPTER 12 - UI_EVENT_MANAGER</b> .....	<b>.155</b>
General Members	
UI_EVENT_MANAGER::UI_EVENT_MANAGER	
UI_EVENT_MANAGER::~~UI_EVENT_MANAGER	
UI_EVENT_MANAGER::Add	
UI_EVENT_MANAGER::Current	
UI_EVENT_MANAGER::DeviceImage	
UI_EVENT_MANAGER::DevicePosition	
UI_EVENT_MANAGER::DeviceState	
UI_EVENT_MANAGER::Event	
UI_EVENT_MANAGER::First	
UI_EVENT_MANAGER::Get	
UI_EVENT_MANAGER::Last	
UI_EVENT_MANAGER::Put	
UI_EVENT_MANAGER::QFlags	
UI_EVENT_MANAGER::Subtract	
UI_EVENT_MANAGER::operator +	
UI_EVENT_MANAGER::operator -	
<b>CHAPTER 13 - UI_EVENT_MAP</b> .....	<b>.175</b>
General Members	
UI_EVENT_MAP::MapEvent	
<b>CHAPTER 14 - UI_GEOMETRY_MANAGER</b> .....	<b>.179</b>
General Members	
UI_GEOMETRY_MANAGER::UI_GEOMETRY_MANAGER	
UI_GEOMETRY_MANAGER::~~UI_GEOMETRY_MANAGER	
UI_GEOMETRY_MANAGER::Add	
UI_GEOMETRY_MANAGER::operator +	
UI_GEOMETRY_MANAGER::ClassName	
UI_GEOMETRY_MANAGER::Current	
UI_GEOMETRY_MANAGER::Event	
UI_GEOMETRY_MANAGER::First	
UI_GEOMETRY_MANAGER::Information	
UI_GEOMETRY_MANAGER::Last	
UI_GEOMETRY_MANAGER::Subtract	
Storage Members	
UI_GEOMETRY_MANAGER::UI_GEOMETRY_MANAGER	
UI_GEOMETRY_MANAGER::Load	
UI_GEOMETRY_MANAGER::New	

UI_GEOMETRY_MANAGER::NewFunction	
UI_GEOMETRY_MANAGER::Store	
<b>CHAPTER 15 - UI_GRAPHICS_DISPLAY</b> . . . . .	<b>195</b>
General Members	
UI_GRAPHICS_DISPLAY::UI_GRAPHICS_DISPLAY	
UI_GRAPHICS_DISPLAY::~~UI_GRAPHICS_DISPLAY	
UI_GRAPHICS_DISPLAY::SetFont	
UI_GRAPHICS_DISPLAY::SetPattern	
<b>CHAPTER 16 - UI_HELP_STUB</b> . . . . .	<b>203</b>
General Members	
UI_HELP_STUB::~~UI_HELP_STUB	
UI_HELP_STUB::DisplayHelp	
<b>CHAPTER 17 - UI_HELP_SYSTEM</b> . . . . .	<b>205</b>
General Members	
UI_HELP_SYSTEM::UI_HELP_SYSTEM	
UI_HELP_SYSTEM::~~UI_HELP_SYSTEM	
UI_HELP_SYSTEM::DisplayHelp	
UIW_HELP_SYSTEM::SetLanguage	
<b>CHAPTER 18 - UI_ITEM</b> . . . . .	<b>213</b>
<b>General Members</b>	
<b>CHAPTER 19 - UI_KEY</b> . . . . .	<b>217</b>
<b>General Members</b>	
<b>CHAPTER 20 - UI_LIST</b> . . . . .	<b>219</b>
General Members	
UI_LIST::UI_LIST	
UI_LIST::~~UI_LIST	
UI_LIST::Add	
UI_LIST::operator +	
UI_LIST::Count	
UI_LIST::Current	
UI_LIST::Destroy	
UI_LIST::First	
UI_LIST::Get	
UI_LIST::Index	
UI_LIST::Last	
UI_LIST::SetCurrent	
UI_LIST::Sort	



UI_LIST::Subtract	
UI_LIST::operator -	
<b>CHAPTER 21 - UI_LIST_BLOCK</b>	<b>237</b>
General Members	
UI_LIST_BLOCK::UI_LIST_BLOCK	
UI_LIST_BLOCK::~~UI_LIST_BLOCK	
UI_LIST_BLOCK::Add	
UI_LIST_BLOCK::Full	
UI_LIST_BLOCK::Subtract	
<b>CHAPTER 22 - UI_MACINTOSH_DISPLAY</b>	<b>245</b>
General Members	
UI_MACINTOSH_DISPLAY::UI_MACINTOSH_DISPLAY	
UI_MACINTOSH_DISPLAY::~~UI_MACINTOSH_DISPLAY	
UI_MACINTOSH_DISPLAY::MapRGB Color	
<b>CHAPTER 23 - UI_MSC_DISPLAY</b>	<b>251</b>
General Members	
UI_MSC_DISPLAY::UI_MSC_DISPLAY	
UI_MSC_DISPLAY::~~UI_MSC_DISPLAY	
UI_MSC_DISPLAY::SetFont	
UI_MSC_DISPLAY::SetPattern	
<b>CHAPTER 24 - UI_MSWINDOWS_DISPLAY</b>	<b>259</b>
General Members	
UI_MSWINDOWS_DISPLAY::UI_MSWINDOWS_DISPLAY	
UI_MSWINDOWS_DISPLAY::~~UI_MSWINDOWS_DISPLAY	
<b>CHAPTER 25 - UI_NEXTSTEP_DISPLAY</b>	<b>265</b>
General Members	
UI_NEXTSTEP_DISPLAY::UI_NEXTSTEP_DISPLAY	
UI_NEXTSTEP_DISPLAY::~~UI_NEXTSTEP_DISPLAY	
UI_NEXTSTEP_DISPLAY::MapNSColor	
<b>CHAPTER 26 - UI_OS2_DISPLAY</b>	<b>271</b>
General Members	
UI_OS2_DISPLAY::UI_OS2_DISPLAY	
UI_OS2_DISPLAY::~~UI_OS2_DISPLAY	
UI_OS2_DISPLAY::SetFont	
<b>CHAPTER 27 - UI_PALETTE</b>	<b>277</b>
General Members	

<b>CHAPTER 28 - UI_PALETTE_MAP</b> .....	279
General Members	
UI_PALETTE_MAP::MapPalette	
<b>CHAPTER 29 - UI_PATH</b> .....	283
General Members	
UI_PATH::UI_PATH	
UI_PATH::~~UI_PATH	
UI_PATH::Current	
UI_PATH::First	
UI_PATH::FirstPathName	
UI_PATH::Last	
UI_PATH::NextPathName	
<b>CHAPTER 30 - UI_PATH_ELEMENT</b> .....	291
General Members	
UI_PATH_ELEMENT::UI_PATH_ELEMENT	
UI_PATH_ELEMENT::~~UI_PATH_ELEMENT	
UI_PATH_ELEMENT::Next	
UI_PATH_ELEMENT::Previous	
<b>CHAPTER 31 - IMPOSITION</b> .....	295
General Members	
UI_POSITION::Assign	
UI_POSITION::operator ==	
UI_POSITION::operator !=	
UI_POSITION::operator <	
UI_POSITION::operator >	
UI_POSITION::operator >=	
UI_POSITION::operator <=	
UI_POSITION::operator ++	
UI_POSITION::operator --	
UI_POSITION::operator +=	
UIPOSITION::operator -=	
<b>CHAPTER 32 - IMPRINTER</b> .....	307
General Members	
UI_PRINTER::UI_PRINTER	
UI_PRINTER::~~UI_PRINTER	
IMPRINTER::BeginPage	
UI_PRINTER::BeginPrintJob	
UI_PRINTER::EndPage	
UI_PRINTER::EndPrintJob	

UI_PRINTER::ScreenDump	
<b>CHAPTER 33 - UI_QUEUE_BLOCK</b> .....	317
General Members	
UI_QUEUE_BLOCK::UI_QUEUE_BLOCK	
UI_QUEUE_BLOCK::~~UI_QUEUE_BLOCK	
UI_QUEUE_BLOCK::Current	
UI_QUEUE_BLOCK::First	
UI_QUEUE_BLOCK::Last	
<b>CHAPTER 34 - UI_QUEUE_ELEMENT</b> .....	323
General Members	
UI_QUEUE_ELEMENT::UI_QUEUE_ELEMENT	
UI_QUEUE_ELEMENT::~~UI_QUEUE_ELEMENT	
UI_QUEUE_ELEMENT::Next	
UI_QUEUE_ELEMENT::Previous	
<b>CHAPTER 35 - UI_REGION</b> .....	327
General Members	
UI_REGION::Assign	
UI_REGION::Encompassed	
UI_REGION::Height	
UI_REGION::Overlap	
UI_REGION::Touching	
UI_REGION::Width	
UI_REGION::operator ==	
UI_REGION::operator !=	
UI_REGION::operator ++	
UI_REGION::operator --	
UI_REGION::operator +=	
UI_REGION::operator -=	
<b>CHAPTER 36 - UI_REGION_ELEMENT</b> .....	341
General Members	
UI_REGION_ELEMENT::UI_REGION_ELEMENT	
UI_REGION_ELEMENT::~~UI_REGION_ELEMENT	
UI_REGION_ELEMENT::Next	
UI_REGION_ELEMENT::Previous	
<b>CHAPTER 37 - UI_REGION_LIST</b> .....	347
General Members	
UI_REGION_LIST::Current	
UI_REGION_LIST::First	

UI_REGION_LIST::Last	
UI_REGION_LIST::Split	
<b>CHAPTER 38 - UI_RELATIVE_CONSTRAINT</b> . . . . .	353
General Members	
UI_RELATIVE_CONSTRAINT::UI_RELATIVE_CONSTRAINT	
UI_RELATIVE_CONSTRAINT::~~UI_RELATIVE_CONSTRAINT	
UI_RELATIVE_CONSTRAINT::Information	
UI_RELATIVE_CONSTRAINT::Modify	
Storage Members	
UI_RELATIVE_CONSTRAINT::UI_RELATIVE_CONSTRAINT	
UI_RELATIVE_CONSTRAINT::Load	
UI_RELATIVE_CONSTRAINT::New	
UI_RELATIVE_CONSTRAINT::NewFunction	
UI_RELATIVE_CONSTRAINT::Store	
<b>CHAPTER 39 - UI_SCROLL_INFORMATION</b> . . . . .	365
General Members	
<b>CHAPTER 40 - UI_TEXT_DISPLAY</b> . . . . .	369
General Members	
UI_TEXT_DISPLAY::UI_TEXT_DISPLAY	
UI_TEXT_DISPLAY::~~UI_TEXT_DISPLAY	
Internationalization Members	
<b>CHAPTER 41 - UI_WCC_DISPLAY</b> . . . . .	375
General Members	
UI_WCC_DISPLAY::UI_WCC_DISPLAY	
UI_WCC_DISPLAY::~~UI_WCC_DISPLAY	
UI_WCC_DISPLAY::SetFont	
UI_WCC_DISPLAY::SetPattern	
<b>CHAPTER 42 - UI_WINDOW_MANAGER</b> . . . . .	383
General Members	
UI_WINDOW_MANAGER::UI_WINDOW_MANAGER	
UI_WINDOW_MANAGER::~~UI_WINDOW_MANAGER	
UI_WINDOW_MANAGER::Add	
UI_WINDOW_MANAGER::operator +	
UI_WINDOW_MANAGER::Center	
UI_WINDOW_MANAGER::Event	
UI_WINDOW_MANAGER::Information	
UI_WINDOW_MANAGER::Subtract	
UI_WINDOW_MANAGER: operator -	

**CHAPTER 43 - UI\_WINDOW\_OBJECT** ..... 399

General Members

UI\_WINDOW\_OBJECT::UI\_WINDOW\_OBJECT  
UI\_WINDOW\_OBJECT::~~UI\_WINDOW\_OBJECT  
UI\_WINDOW\_OBJECT::ClassName  
UI\_WINDOW\_OBJECT::CreateMotifString  
UI\_WINDOW\_OBJECT::DrawBorder  
UI\_WINDOW\_OBJECT::DrawItem  
UI\_WINDOW\_OBJECT::DrawShadow  
UI\_WINDOW\_OBJECT::DrawText  
UI\_WINDOW\_OBJECT::Event  
UI\_WINDOW\_OBJECT::Font  
UI\_WINDOW\_OBJECT::Get  
UI\_WINDOW\_OBJECT::HotKey  
UI\_WINDOW\_OBJECT::Information  
UI\_WINDOW\_OBJECT::Inherited  
UI\_WINDOW\_OBJECT::LogicalEvent  
UI\_WINDOW\_OBJECT::LogicalPalette  
UI\_WINDOW\_OBJECT::Modify  
UI\_WINDOW\_OBJECT::NeedsUpdate  
UI\_WINDOW\_OBJECT::Next  
UI\_WINDOW\_OBJECT::NumberID  
UI\_WINDOW\_OBJECT::Previous  
UI\_WINDOW\_OBJECT::RedisplayType  
UI\_WINDOW\_OBJECT::RegionConvert  
UI\_WINDOW\_OBJECT::RegionMax  
UI\_WINDOW\_OBJECT::RegisterObject  
UI\_WINDOW\_OBJECT::Root  
UI\_WINDOW\_OBJECT::SearchID  
UI\_WINDOW\_OBJECT::StringID  
UI\_WINDOW\_OBJECT::TopWidget  
UI\_WINDOW\_OBJECT::UserFunction  
UI\_WINDOW\_OBJECT::Validate

Storage Members

UI\_WINDOW\_OBJECT::UI\_WINDOW\_OBJECT  
UI\_WINDOW\_OBJECT::Load  
UI\_WINDOW\_OBJECT::New  
UI\_WINDOW\_OBJECT::NewFunction  
UI\_WINDOW\_OBJECT::Store

**CHAPTER 44 - UI\_XT\_DISPLAY** ..... 459

General Members

UI\_XT\_DISPLAY::UI\_XT\_DISPLAY

UI\_XT\_DISPLAY::~~UI\_XT\_DISPLAY

**CHAPTER 45 - UID\_CURSOR** . . . . . 465

General Members  
UID\_CURSOR::UID\_CURSOR  
UID\_CURSOR::~~UID\_CURSOR  
UID\_CURSOR::Event  
UID\_CURSOR::Poll

**CHAPTER 46 - UID\_KEYBOARD** . . . . . 473

General Members  
UID\_KEYBOARD::UID\_KEYBOARD  
UID\_KEYBOARD::~~UID\_KEYBOARD  
UID\_KEYBOARD::Event  
UID\_KEYBOARD::Poll

**CHAPTER 47 - UID\_MOUSE** . . . . . 481

General Members  
UID\_MOUSE::UID\_MOUSE  
UID\_MOUSE::~~UID\_MOUSE  
UID\_MOUSE::Event  
UID\_MOUSE::MouseMove  
UID\_MOUSE::Poll  
Internationalization Members

**CHAPTER 48 - UID\_TIMER** . . . . . 493

General Members  
UID\_TIMER::UID\_TIMER  
UID\_TIMER::~~UID\_TIMER  
UID\_TIMER::Event  
UID\_TIMER::Poll

**CHAPTER 49 - ZIL\_BIGNUM** . . . . . 499

General Members  
ZIL\_BIGNUM::ZIL\_BIGNUM  
ZIL\_BIGNUM::~~ZIL\_BIGNUM  
ZIL\_BIGNUM::abs  
ZIL\_BIGNUM::ceil  
ZIL\_BIGNUM::Export  
ZIL\_BIGNUM::floor  
ZIL\_BIGNUM::GetLocale  
ZIL\_BIGNUM::Import  
ZIL\_BIGNUM::round

ZIL\_BIGNUM::SetLocale  
 ZIL\_BIGNUM::truncate  
 ZIL\_BIGNUM::operator =  
 ZIL\_BIGNUM::operator +  
 ZIL\_BIGNUM::operator -  
 ZIL\_BIGNUM::operator \*  
 ZIL\_BIGNUM::operator ++  
 ZIL\_BIGNUM::operator --  
 ZIL\_BIGNUM::operator +=  
 ZIL\_BIGNUM::operator -=  
 ZIL\_BIGNUM::operator ==  
 ZIL\_BIGNUM::operator !=  
 ZIL\_BIGNUM::operator >  
 ZIL\_BIGNUM::operator >=  
 ZIL\_BIGNUM::operator <  
 ZIL\_BIGNUM::operator <=

**CHAPTER 50 - ZIL\_BITMAP\_ELEMENT** ..... 527

General Members

**CHAPTER 51 - ZIL\_DATE** ..... 529

General Members

ZIL\_DATE::ZIL\_DATE  
 ZIL\_DATE::DayOfWeek  
 ZIL\_DATE::DaysInMonth  
 ZIL\_DATE::DaysInYear  
 ZIL\_DATE::Export  
 ZIL\_DATE::GetBasis  
 ZIL\_DATE::Import  
 ZIL\_DATE::SetBasis  
 ZIL\_DATE::operator =  
 ZIL\_DATE::operator +  
 ZIL\_DATE::operator -  
 ZIL\_DATE::operator >  
 ZIL\_DATE::operator >=  
 ZIL\_DATE::operator <  
 ZIL\_DATE::operator <=  
 ZIL\_DATE::operator ++  
 ZIL\_DATE::operator -  
 ZIL\_DATE::operator +=  
 ZIL\_DATE::operator -=  
 ZIL\_DATE::operator ==  
 ZIL\_DATE::operator !=

<b>CHAPTER 52 - ZIL_DECORATION</b> . . . . .	557
General Members	
ZIL_DECORATION::ZIL_DECORATION	
ZIL_DECORATION::AssignData	
ZIL_DECORATION::DeleteData	
ZIL_DECORATION::GetBitmap	
ZIL_DECORATION::GetText	
Storage Members	
ZIL_DECORATION::ClassLoadData	
ZIL_DECORATION::ClassStoreData	
<b>CHAPTER 53 - ZIL_DECORATION_MANAGER</b> . . . . .	563
General Members	
ZIL_DECORATION_MANAGER::ZIL_DECORATION_MANAGER	
ZIL_DECORATION_MANAGER::CreateData	
ZIL_DECORATION_MANAGER::FreeDecorations	
ZIL_DECORATION_MANAGER::LoadDefaultDecorations	
ZIL_DECORATION_MANAGER::SetDecorations	
ZIL_DECORATION_MANAGER::UseDecorations	
<b>CHAPTER 54 - ZIL_DELTA_STORAGE_OBJECT</b> . . . . .	569
General Members	
ZIL_DELTA_STORAGE_OBJECT::ZIL_DELTA_STORAGE_OBJECT	
ZIL_DELTA_STORAGE_OBJECT::~ZIL_DELTA_STORAGE_OBJECT	
ZIL_DELTA_STORAGE_OBJECT::Store	
<b>CHAPTER 55 -</b>	
<b>ZIL_DELTA_STORAGE_OBJECT_READ_ONLY</b> . . . . .	575
General Members	
ZIL_DELTA_STORAGE_OBJECT_READ_ONLY::ZIL_DELTA_	
STORAGE_OBJECT_READ_ONLY	
ZIL_DELTA_STORAGE_OBJECT_READ_ONLY::~ZIL_DELTA_	
STORAGE_OBJECT_READ_ONLY	
ZIL_DELTA_STORAGE_OBJECT_READ_ONLY::Load	
<b>CHAPTER 56 - ZIL_I18N</b> . . . . .	581
General Members	
ZIL_I18N::ZIL_I18N	
ZIL_I18N::~ZIL_I18N	
ZIL_I18N::AssignData	
ZIL_I18N::DeleteData	
Storage Members	
ZIL_I18N::ClassLoadData	



ZIL\_I18N::ClassStoreData  
ZIL\_I18N::Load  
ZIL\_I18N::Store  
ZIL\_I18N::Traverse

**CHAPTER 57 - ZIL\_I18N\_MANAGER** . . . . . 589

General Members  
ZIL\_I18N\_MANAGER::CreateData  
ZIL\_I18N\_MANAGER::FreeI18N  
ZIL\_I18N\_MANAGER::LoadDefaultI18N  
ZIL\_I18N\_MANAGER::UseI18N

**CHAPTER 58 - ZIL\_INTERNATIONAL** . . . . . 593

General Members  
ZIL\_INTERNATIONAL::CharMapInitialize  
ZIL\_INTERNATIONAL::chartod  
ZIL\_INTERNATIONAL::ConvertFromFilename  
ZIL\_INTERNATIONAL::ConvertToFilename  
ZIL\_INTERNATIONAL::DecomposeCharacter  
ZIL\_INTERNATIONAL::DecomposeString  
ZIL\_INTERNATIONAL::DefaultI18nInitialize  
ZIL\_INTERNATIONAL::I18nInitialize  
ZIL\_INTERNATIONAL::IsNonSpacing  
ZIL\_INTERNATIONAL::ISOtoICHAR  
ZIL\_INTERNATIONAL::ISOtoUNICODE  
ZIL\_INTERNATIONAL::LoadICHARtoHardware  
ZIL\_INTERNATIONAL::MapChar  
ZIL\_INTERNATIONAL::MapText  
ZIL\_INTERNATIONAL::mblen  
ZIL\_INTERNATIONAL::mbstowes  
ZIL\_INTERNATIONAL::OSI18nInitialize  
ZIL\_INTERNATIONAL::StripHotMark  
ZIL\_INTERNATIONAL::strstrip  
ZIL\_INTERNATIONAL::TimeStamp  
ZIL\_INTERNATIONAL::UnMapChar  
ZIL\_INTERNATIONAL::UnMapText  
ZIL\_INTERNATIONAL::westombs  
ZIL\_INTERNATIONAL::WildStrcmp  
Internationalization Members  
ZIL\_INTERNATIONAL::MachineName  
ZIL\_INTERNATIONAL::ParseLangEnv

<b>CHAPTER 59 - ZIL_LANGUAGE</b> .....	<b>617</b>
General Members	
ZIL_LANGUAGE::ZIL_LANGUAGE	
ZIL_LANGUAGE::AssignData	
ZIL_LANGUAGE::DeleteData	
ZIL_LANGUAGE::GetMessage	
Storage Members	
ZIL_LANGUAGE::ZIL_LANGUAGE	
ZIL_LANGUAGE::ClassLoadData	
ZIL_LANGUAGE::ClassStoreData	
ZIL_LANGUAGE::Load	
ZIL_LANGUAGE::Store	
<b>CHAPTER 60 - ZIL_LANGUAGE_ELEMENT</b> .....	<b>625</b>
General Members	
ZIL_LANGUAGE_ELEMENT::SwapData	
<b>CHAPTER 61 - ZIL_LANGUAGE_MANAGER</b> .....	<b>627</b>
General Members	
ZIL_LANGUAGE_MANAGER::ZIL_LANGUAGE_MANAGER	
ZIL_LANGUAGE_MANAGER::CreateData	
ZIL_LANGUAGE_MANAGER::FreeLanguage	
ZIL_LANGUAGE_MANAGER::LoadDefaultLanguage	
ZIL_LANGUAGE_MANAGER::SetLanguage	
ZIL_LANGUAGE_MANAGER::UseLanguage	
<b>CHAPTER 62 - ZIL_LOCALE</b> .....	<b>633</b>
General Members	
ZIL_LOCALE::ZIL_LOCALE	
ZIL_LOCALE::AssignData	
ZIL_LOCALE::DeleteData	
Storage Members	
ZIL_LOCALE::ClassLoadData	
ZIL_LOCALE::ClassStoreData	
<b>CHAPTER 63 - ZIL_LOCALE_ELEMENT</b> .....	<b>639</b>
General Members	
<b>CHAPTER 64 - ZIL_LOCALE_MANAGER</b> .....	<b>643</b>
General Members	
ZIL_LOCALE_MANAGER::ZIL_LOCALE_MANAGER	
ZIL_LOCALE_MANAGER::CreateData	
ZIL_LOCALE_MANAGER::FreeLocale	

ZIL_LOCALE_MANAGER::LoadDefaultLocale	
ZIL_LOCALE_MANAGER::SetLocale	
ZIL_LOCALE_MANAGER::UseLocale	
<b>CHAPTER 65 - ZIL_MAP_CHARS</b> . . . . .	<b>649</b>
General Members	
ZIL_MAP_CHARS::ZIL_MAP_CHARS	
ZIL_MAP_CHARS::~ZIL_MAP_CHARS	
ZIL_MAP_CHARS::MapChar	
ZIL_MAP_CHARS::MapText	
ZIL_MAP_CHARS::mblen	
ZIL_MAP_CHARS::mbstowcs	
ZIL_MAP_CHARS::UnMapChar	
ZIL_MAP_CHARS::UnMapText	
ZIL_MAP_CHARS::wctombs	
<b>CHAPTER 66 - ZIL_STORAGE</b> . . . . .	<b>659</b>
General Members	
ZIL_STORAGE::ZIL_STORAGE	
ZIL_STORAGE::~ZIL_STORAGE	
ZIL_STORAGE::DestroyObject	
ZIL_STORAGE::Flush	
ZIL_STORAGE::Link	
ZIL_STORAGE::MkDir	
ZIL_STORAGE::RenameObject	
ZIL_STORAGE::Rmdir	
ZIL_STORAGE::Save	
ZIL_STORAGE::SaveAs	
<b>CHAPTER 67 - ZIL_STORAGE_DIRECTORY</b> . . . . .	<b>671</b>
General Members	
ZIL_STORAGE_DIRECTORY::~ZIL_STORAGE_DIRECTORY	
ZIL_STORAGE_DIRECTORY::ReadDir	
ZIL_STORAGE_DIRECTORY::RewindDir	
ZIL_STORAGE_DIRECTORY::SeekDir	
ZIL_STORAGE_DIRECTORY::TellDir	
<b>CHAPTER 68 - ZIL_STORAGE_OBJECT</b> . . . . .	<b>675</b>
General Members	
ZIL_STORAGE_OBJECT::ZIL_STORAGE_OBJECT	
ZIL_STORAGE_OBJECT::~ZIL_STORAGE_OBJECT	
ZIL_STORAGE_OBJECT::SetCTime	
ZIL_STORAGE_OBJECT::SetMTime	

ZIL\_STORAGE\_OBJECT::Store  
ZIL\_STORAGE\_OBJECT::Touch

**CHAPTER 69 -**

**ZIL\_STORAGE\_OBJECT\_READ\_ONLY** . . . . . 683

General Members

ZIL\_STORAGE\_OBJECT\_READ\_ONLY::ZIL\_STORAGE\_OBJECT\_READ\_ONLY  
ZIL\_STORAGE\_OBJECT\_READ\_ONLY::~ZIL\_STORAGE\_ -  
OBJECT\_READ\_ONLY  
ZIL\_STORAGE\_OBJECT\_READ\_ONLY::Load  
ZIL\_STORAGE\_OBJECT\_READ\_ONLY::Seek  
ZIL\_STORAGE\_OBJECT\_READ\_ONLY::Stats  
ZIL\_STORAGE\_OBJECT\_READ\_ONLY::Storage  
ZIL\_STORAGE\_OBJECT\_READ\_ONLY::Store  
ZIL\_STORAGE\_OBJECT\_READ\_ONLY::Tell

**CHAPTER 70 - ZIL\_STORAGE\_READ\_ONLY** . . . . . 693

General Members

ZIL\_STORAGE\_READ\_ONLY::ZIL\_STORAGE\_READ\_ONLY  
ZIL\_STORAGE\_READ\_ONLY::~ZIL\_STORAGE\_READ\_ONLY  
ZIL\_STORAGE\_READ\_ONLY::AppendFullPath  
ZIL\_STORAGE\_READ\_ONLY::ChangeExtension  
ZIL\_STORAGE\_READ\_ONLY::ChDir  
ZIL\_STORAGE\_READ\_ONLY::FindFirstID  
ZIL\_STORAGE\_READ\_ONLY::FindFirstObject  
ZIL\_STORAGE\_READ\_ONLY::FindNextID  
ZIL\_STORAGE\_READ\_ONLY::FindNextObject  
ZIL\_STORAGE\_READ\_ONLY::GetCWD  
ZIL\_STORAGE\_READ\_ONLY::MakeFullPath  
ZIL\_STORAGE\_READ\_ONLY::OpenDir  
ZIL\_STORAGE\_READ\_ONLY::Stats  
ZIL\_STORAGE\_READ\_ONLY::StorageName  
ZIL\_STORAGE\_READ\_ONLY::StripFullPath  
ZIL\_STORAGE\_READ\_ONLY::TempName  
ZIL\_STORAGE\_READ\_ONLY::ValidName  
ZIL\_STORAGE\_READ\_ONLY::Version

**CHAPTER 71 - ZIL\_TEXT\_ELEMENT** . . . . . 713

General Members

**CHAPTER 72 - ZIL\_TIME** . . . . . 715

General Members

ZIL\_TIME::ZIL\_TIME

ZIL\_TIME::Export  
 ZIL\_TIME::Import  
 ZIL\_TIME::operator =  
 ZIL\_TIME::operator +  
 ZIL\_TIME::operator -  
 ZIL\_TIME::operator >  
 ZIL\_TIME::operator >=  
 ZIL\_TIME::operator <  
 ZIL\_TIME::operator <=  
 ZIL\_TIME::operator ++  
 ZIL\_TIME::operator --  
 ZIL\_TIME::operator +=  
 ZIL\_TIME::operator -=  
 ZIL\_TIME::operator ==  
 ZIL\_TIME::operator !=

**CHAPTER 73 - ZIL\_UTIME** .....739

General Members  
 ZIL\_UTIME::ZIL\_UTIME  
 ZIL\_UTIME::~ZIL\_UTIME  
 ZIL\_UTIME::ConvertJday  
 ZIL\_UTIME::ConvertUsec  
 ZIL\_UTIME::DayOfWeek  
 ZIL\_UTIME::DaysInMonth  
 ZIL\_UTIME::DaysInYear  
 ZIL\_UTIME::Export  
 ZIL\_UTIME::Import  
 ZIL\_UTIME::LeapYear  
 ZIL\_UTIME::MakeCanonical  
 ZIL\_UTIME::SetLanguage  
 ZIL\_UTIME::SetLocale  
 ZIL\_UTIME::operator =  
 ZIL\_UTIME::operator +  
 ZIL\_UTIME::operator -  
 ZIL\_UTIME::operator >  
 ZIL\_UTIME::operator >=  
 ZIL\_UTIME::operator <  
 ZIL\_UTIME::operator <=  
 ZIL\_UTIME::operator ==  
 ZIL\_UTIME::operator !=

**INDEX** .....763

# INTRODUCTION

The *Programmer's Reference Volume 1* contains descriptions of OpenZinc Application Framework support 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. Support objects are those objects that are not window objects.

## UI\_SAMPLE\_CLASS::SampleFunction

### Syntax

*returnValue* SampleFunction(*type1 parameter!*, *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*<sub>out</sub> 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*<sub>in</sub> 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*<sub>in/out</sub> 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 UID_KEYBOARD
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 UIWJTIME
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
// GeneralOpenZincSwitches
// 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_VOIDF, 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, 3.5, 3.0 compatibility
```

### ULGEN.HPP

```
#if !defined(UI_GEN_HPP)
#   define UI_GEN_HPP
#   if !defined(UI_ENV_HPP)
#       include <ul_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_UTCIME
// 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
```

## ULDSP.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
```

## UI\_EVT.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 for WOF_BORDER flag option
// UIW_WINDOW
// WNF_FLAGS
// UIW_WINDOW::ZIL_INFO_REQUEST
// UI_WINDOW_MANAGER
// UIW_BORDER
// BDF_FLAGS
// UIW_PROMPT
// UIW_BUTTON
// BTF_FLAGS
// BTS_STATUS
// UIW_BUTTON::ZIL_INFO_REQUEST
// UIW_TITLE

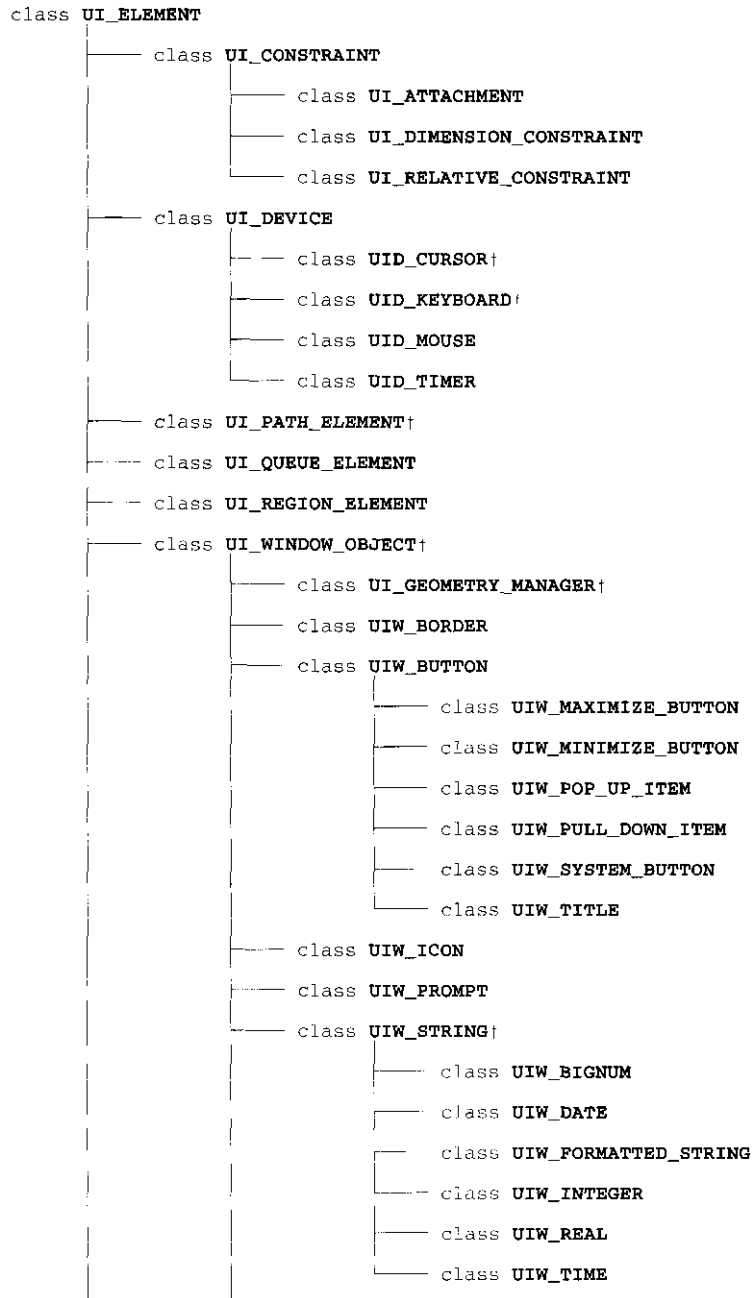
```

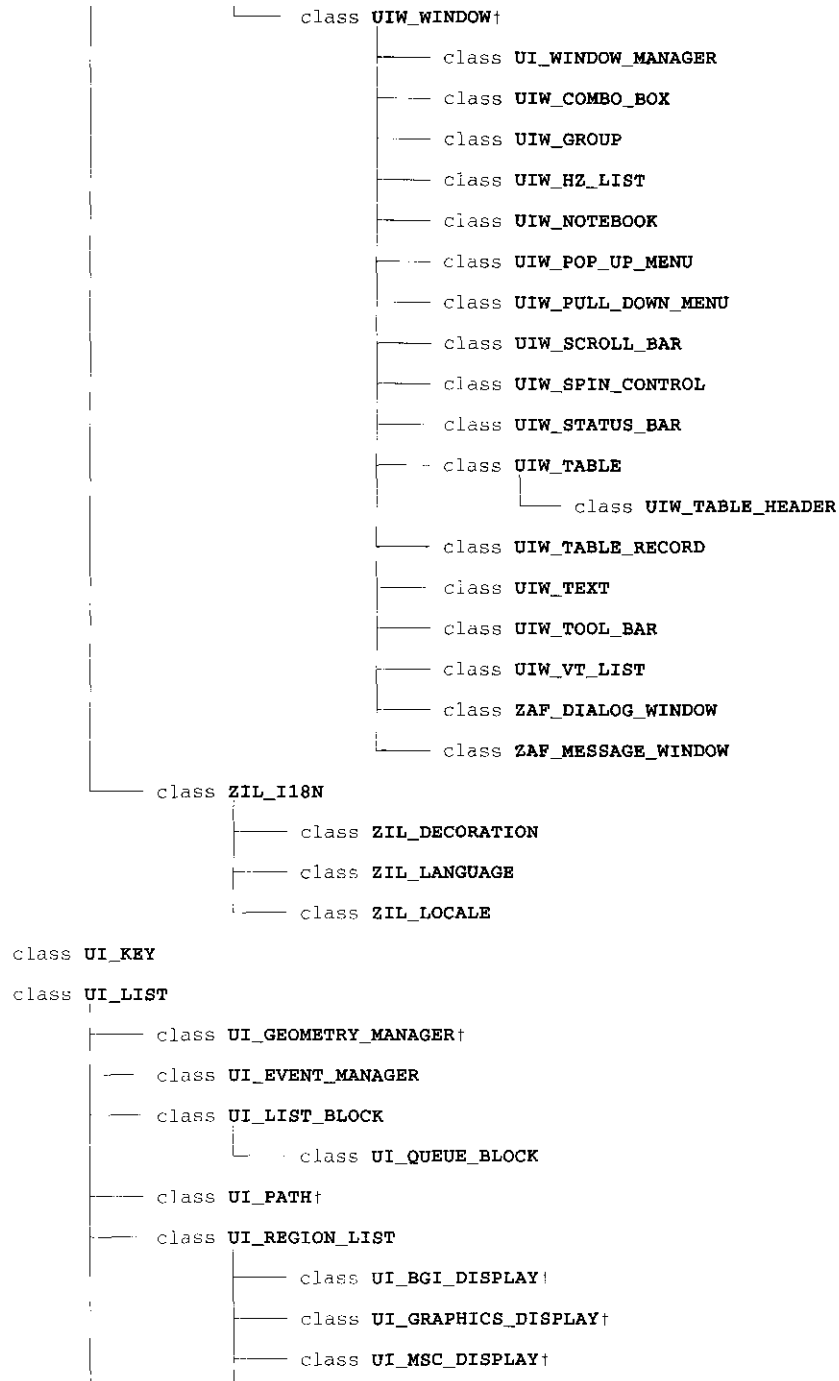
```

// 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_LIST
// UIW_HZ_LIST
// UIW_COMBO_BOX
// UIW_COMBO_BOX::ZIL_INFO_REQUEST
// UIW_SPIN_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_RELATIVE_CONSTRAINT
// RLCF_FLAG
// UI_RELATIVE_CONSTRAINT::ZIL_INFO_REQUEST
// UI_GEOMETRY_MANAGER
// Message indexes for the help and error system windows.
// Version 3.6 compatibility

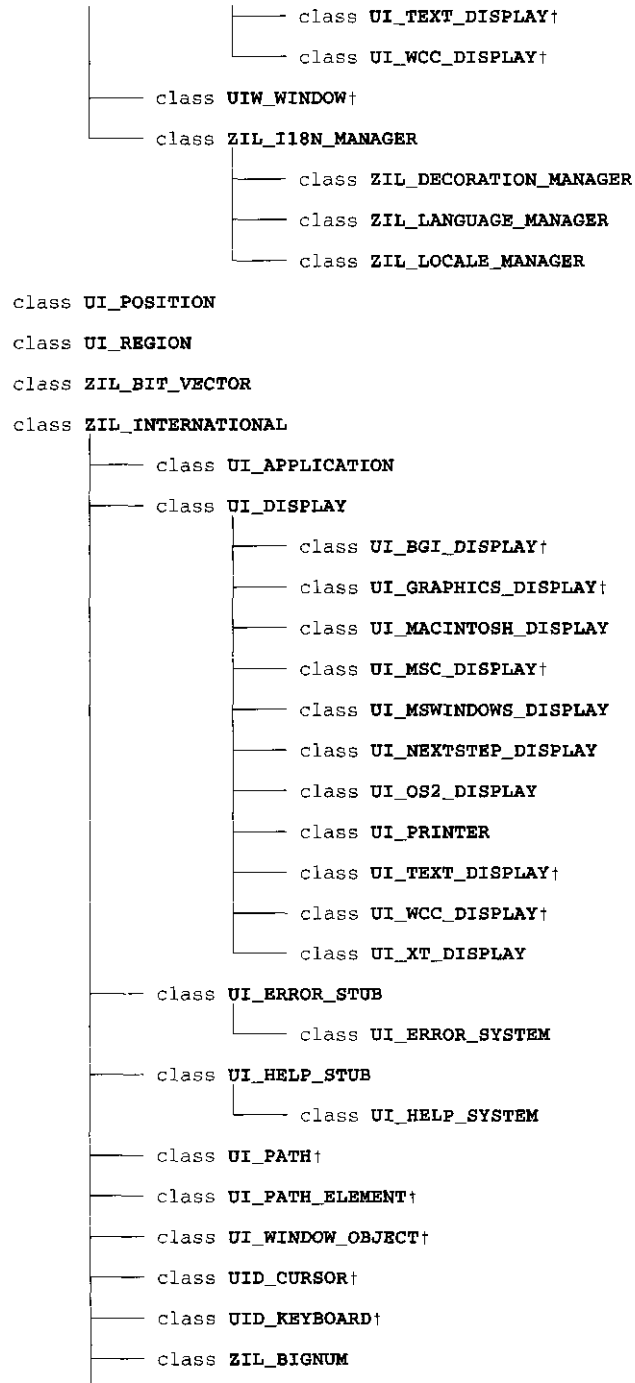
```

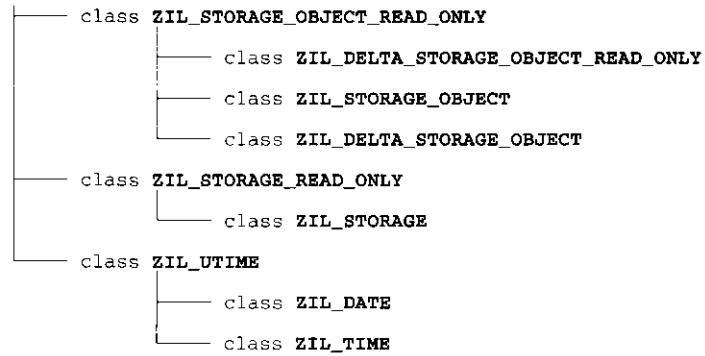
## CLASS HIERARCHY











```

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 - UI\_APPLICATION

The `UI_APPLICATION` class is used to initialize the standard control objects for an application built with OpenZinc Application Framework. The class sets up the display, the Event Manager and the Window Manager, and also provides a `main( )` function (or `WinMain( )` for Windows and Windows NT), removing these responsibilities from the programmer. This provides for a clean initialization module that is completely portable across platforms. Use of this class is optional (except for NEXTSTEP applications—discussed below), but if it is used, the programmer merely has to provide application-specific initialization and the main control loop to retrieve and dispatch events.

Application-specific initialization is performed in the `UI_APPLICATION::Main( )` function. The definition of `UI_APPLICATION::Main()` must be provided by; the programmer if the `UI_APPLICATION` class is to be used. The reference to the `UI_APPLICATION` class when defining this function causes a `main()` (or `WinMain()`) function to be linked in with the program. This `main()` function creates an instance of `UI_APPLICATION` and calls the programmer-defined `UI_APPLICATION::Main()`.

**NOTE:** It is important that one, and only one, definition of `UI_APPLICATION::Main( )` is provided and that a definition is provided only if using the `UI_APPLICATION` class.

If this class is not used, the `main()` (or `WinMain()`) function must still be provided by the programmer, as in any C++ program. The display, Event Manager and Window Manager must then also be created manually.

**NOTE:** If the `UI_APPLICATION` class is not used, no reference should be made to the class, since the linker will then attempt to link in another `main( )`, resulting in a linker error.

Use of the `UI_APPLICATION` class is not required except for NEXTSTEP applications. Due to the event handling requirements of both OpenZinc Application Framework and NEXTSTEP, their interaction is not straightforward. The `UI_APPLICATION` class handles this interaction properly.

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

```
class ZIL_EXPORT_CLASS UI_APPLICATION : public ZIL_INTERNATIONAL
{
public:
    UI_DISPLAY *display;
    UI_EVENT_MANAGER *eventManager;
    UI_WINDOW_MANAGER *windowManager;
    UI_PATH *searchPath;
```

```

#if defined(ZIL_NEXTSTEP)
    DPSTimedEntry myTimedEvent;
#endif
#if defined(ZIL_MSWINDOWS)
    HANDLE hlnstance;
    HANDLE hPrevInstance;
    LPSTR lpszCmdLine;
    int nCmdShow;
    UI_APPLICATION(HANDLE hlnstance, HANDLE hPrevInstance,
        LPSTR lpszCmdLine, int nCmdShow);
#else
    UI_APPLICATION(int argc, char **argv);
#endif
int argc;
ZIL_ICHAR **argv;

~UI_APPLICATION(void);
int Main(void);
EVENT_TYPE Control(void);
void LinkMain(void);

public:
    static ZIL_LANGUAGE_ELEMENT *_textSwitches;
    static ZIL_LANGUAGE_ELEMENT *_graphicSwitches;
};

```

## General Members

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

- *display* is a pointer to the UI\_DISPLAY that is created in the UI\_APPLICATION constructor. The type of display created depends on which environment is being used. For DOS, the type of display created also depends on which graphics library is being used for the application. For example, if the DOS GFX library is used, *display* will be of type UI\_GRAPHICS\_DISPLAY. See "Appendix A—Compiler Considerations" in the *Getting Started* manual for more information on using graphics libraries in OpenZinc programs, *display* will be of type UI\_MACINTOSH\_DISPLAY if the application is a Macintosh application, UI\_MSWINDOWS\_DISPLAY if the application is a Windows application, UI\_OS2\_DISPLAY for an OS/2 application, UI\_XT\_DISPLAY for a Motif application and UI\_NEXTSTEP\_DISPLAY for a NEXTSTEP application.
- *eventManager* is a pointer to the Event Manager created in the UI\_APPLICATION constructor.
- *windowManager* is a pointer to the Window Manager created in the UI\_APPLICATION constructor.
- *searchPath* is a pointer to a UI\_PATH object containing the program startup directory as supplied by argv[0], if the application is for DOS, OS/2 or Motif. *searchPath* will

also ensure that the current working directory is searched if access to files is required from within the program. If the application is a Windows application, *searchPath* does not maintain a pointer to the program startup directory but ensures that the current working directory is searched.

- *myTimedEvent* is the tag for the timed entry created in the **Control( )** function in a NEXTSTEP application. In NEXTSTEP applications, a timer is set up that calls a function at frequent intervals. This provides the main event loop for the OpenZinc program that allows the application to process events. *myTimedEvent* is the tag that identifies the timer created.
- *hInstance* is the instance handle of the application. This member is available only in Windows applications.
- *hPrevInstance* is the handle of another instance of the application, if one is running. This member is available only in Windows applications.
- *IpszCmdLine* is a pointer to the string entered at the command line. This member is available only in Windows applications.
- *nCmdShow* indicates how the application is to be displayed upon execution. This member is available only in Windows applications.
- *argc* is a count of the number of command-line arguments that were entered when running the application. This member is available only in non-Windows applications.
- *argv* is a pointer to an array of the command-line arguments that were entered when running the application. This member is available only in non-Windows applications.
- *\_textSwitches* is a collection of strings that can be used as command-line arguments to cause the application to come up in text mode. The UI\_APPLICATION class uses the strings maintained by *JextSwitches* to compare against any command-line arguments in order to determine if the user wanted to run the application in text mode. By default, the only text switch is "/text." This variable is used in DOS mode only.
- *\_graphicSwitches* is a collection of strings that can be used as command-line arguments to cause the application to come up in a particular graphics mode. The UI\_APPLICATION class uses the strings maintained by *\_graphicSwitches* to compare against any command-line arguments in order to determine if the user wanted to run the application in a certain graphics mode. By default, the only graphics switch is

"/svga." This variable is used in DOS mode with the UI\_GRAPHICS\_DISPLAY only.

## **UI\_APPLICATION::UI\_APPLICATION**

### **Syntax**

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

```
UI_APPLICATION(HANDLE hInstance, HANDLE hPrevInstance, LPSTR IpszCmdLine,  
int nCmdShow);  
or
```

```
UI_APPLICATION(int argc, char **argv);
```

### **Portability**

This function is available on the following environments:

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

### **Remarks**

These overloaded advanced constructors create a new UI\_APPLICATION class object. The constructor initializes the *display*, *eventManager*, *windowManager* and *searchPath* member variables. By default, *eventManager* will have a UID\_KEYBOARD, a UID\_MOUSE and a UID\_CURSOR device attached to it.

These constructors should not be called by the programmer but are called from within the UI\_APPLICATION class' **main( )** (or **WinMain( )**) function, which is linked in automatically when a reference to the UI\_APPLICATION class is made.

The first constructor is specific to a Windows or Windows NT application. It takes the following arguments:

- *hInstance<sub>in</sub>* is the instance handle of the application. This parameter is automatically passed in to the application as a **WinMain()** parameter.

- *hPrevInstance<sub>in</sub>* is the handle of another instance of the application, if one is running. This parameter is automatically passed in to the application as a **WinMain( )** parameter.
- *IpszCmdLine<sub>in</sub>* is a pointer to the string entered at the command line. This parameter is automatically passed in to the application as a **WinMain( )** parameter.
- *nCmdShow<sub>in</sub>* indicates if the application's initial window should display in a maximized state, a normal state or a minimized state upon execution of the application. This parameter is automatically passed in to the application as a **WinMain( )** parameter.

The second constructor is specific to non-Windows applications. It takes the following arguments:

- *argc<sub>in</sub>* is a count of the number of command-line arguments that were entered when running the application. This parameter is passed in to the application as a **main( )** parameter.
- *argv<sub>in</sub>* is a pointer to an array of the command-line arguments that were entered when running the application. This parameter is passed in to the application as a **main( )** parameter.

## Example

```
#include <ui_win.hpp>

// Referencing UI_APPLICATION causes a main() or
// WinMain() to be linked in automatically. This main() creates an
// instance of UI_APPLICATION and calls UI_APPLICATION::Main(),
// defined by the programmer.

// This main() is part of the UI_APPLICATION class.
int main(int argc, char **argv)
{
    UI_APPLICATION *application = new UI_APPLICATION(argc, argv);

    // Call the application program,
    int ccode = application->Main();

    // Restore the system,
    delete application;

    return(ccode);
}
```



## UI\_APPLICATION::~~UI\_APPLICATION

### Syntax

```
#include <ui_win.hpp>

~UI_APPLICATION(void);
```

### Portability

This function is available on the following environments:

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

### Remarks

This destructor destroys the *display*, *eventManager*, *windowManager* and *searchPath* members, if they exist. It is not recommended that the programmer modify these members; if they are changed, however, it is important to set these members to NULL if they are deleted prior to the destructor being called.

## UI\_APPLICATION::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 acts as the main control loop used to get events from the Event Manager and to dispatch them to the Window Manager. Use of this function is optional. If this function is not used, the programmer must implement a loop that collects events from the Event Manager and passes them to the Window Manager.

- *returnValue<sub>out</sub>* is the event type that caused the event loop to exit (i.e., L\_EXIT or S\_NO\_OBJECT).

## Example

```
#include <ui_win.hpp>

// Referencing UI_APPLICATION causes a main() or
// WinMain() to be linked in automatically. This main() creates an
// instance of UI_APPLICATION and calls UI_APPLICATION::Main(),
// defined by the programmer.

int UI_APPLICATION::Main(void)
{
    // The UI_APPLICATION constructor automatically initializes the
    // display, eventManager, and windowManager variables.

    // 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;

    // Use Control() to get and dispatch events.
    EVENT_TYPE ccode = Control();

    // DO NOT delete the display, eventManager, or windowManager.
    // They are deleted in the UI_APPLICATION destructor.

    // Return the exit code,
    return (0);
}
```

## UI\_APPLICATION::LinkMain

### Syntax

```
#include <ui_win.hpp>

void LinkMain(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 a stub. Some linkers will not link in a module unless a function in that module is called. Calling the **LinkMain()** function ensures that the module is linked in.

## Example

```
#include <ui_win.hpp>

// Referencing UI_APPLICATION causes a main() or
// WinMain() to be linked in automatically. This main() creates an
// instance of UI_APPLICATION and calls UI_APPLICATION::Main(),
// defined by the programmer.

int UI_APPLICATION::Main(void)
{
    // The UI_APPLICATION constructor automatically initializes the
    // display, eventManager, and windowManager variables.

    LinkMain();

    // 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;

    EVENT_TYPE ccode = Control();

    // DO NOT delete the display, eventManager, or windowManager.
    // They are deleted in the UI_APPLICATION destructor.

    // Return the exit code,
    return (0);
}
```

## UI\_APPLICATION::Main

### Syntax

```
#include <ui_win.hpp>

int Main(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 set up application-specific initialization and the main control loop. It is defined by the programmer. The **main()** (or **WinMain()**) function provided with the `UI_APPLICATION` class calls this function after first creating an instance of `UI_APPLICATION`. It is possible to modify the *display*, *eventManager*, *windowManager* and *searchPath* member variables from within the **Main()** function, but it is not recommended. If these members are modified, it is important to consider the order in which the members are modified, as some of these members maintain pointers to the other members.

NOTE: The programmer must provide one, and only one, definition for this function if **main()** or **WinMain()** is not used. If no definition is provided, or if more than one definition is provided, a linker error will occur.

- *returnValue<sub>out</sub>* is the program exit code.

### Example

```
#include <ui_win.hpp>

// Referencing UI_APPLICATION causes a main() or
// WinMain() to be linked in automatically. This main() creates an
// instance of UI_APPLICATION and calls UI_APPLICATION::Main(),
// defined by the programmer.

int UI_APPLICATION::Main(void)
{
    // The UI_APPLICATION constructor automatically initializes the
    // display, eventManager, and windowManager variables.
```

```

// 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;

EVENT_TYPE ccode = Control();

//DO NOT delete the display, eventManager, or windowManager.
// They are deleted in the UI_APPLICATION destructor.

// Return the exit code.
return (0);

```

## CHAPTER 2 - UI\_ATTACHMENT

The UI\_ATTACHMENT class object is used for geometry management. Specifically, this class allows a managed object to be tied to an edge of its parent or to an edge of a sibling object. The UI\_ATTACHMENT is added to the parent object's geometry manager. See "Chapter 14—UI\_GEOMETRY\_MANAGER" for more details on using the geometry manager.

The UI\_ATTACHMENT class is declared in UI\_WIN.HPP. Its public and protected members are:

```
class UI_ATTACHMENT : public UI_CONSTRAINT
{
public:
    UI_ATTACHMENT(UI_WINDOW_OBJECT * object,
        ATCF_FLAGS atcFlags = ATCF_NO_FLAGS, int offset = 0);
    UI_ATTACHMENT(UI_WINDOW_OBJECT * object, UI_WINDOW_OBJECT preference,
        ATCF_FLAGS atcFlags, int offset = 0);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);
    virtual void Modify(void);

#ifdef 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));
    UI_ATTACHMENT(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

#ifdef 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 *reference;
    ZIL_NUMBERID refNumberID;
    ATCF_FLAGS atcFlags;
    int offset;
};
```

### General Members

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

- *reference* is the sibling object to which the managed object is tied.
- *refNumberID* is the numberID of the reference object to which the attachment is tied.
- *atcFlags* are flags that define the operation of the UI\_ATTACHMENT class. A full description of the attachment flags is given in the UI\_ATTACHMENT constructor.
- *offset* is how far the managed object should be positioned from the object to which it is tied. This value is specified in cell dimensions.

## UI\_ATTACHMENT::UI\_ATTACHMENT

### Syntax

```
#include <ui_win.hpp>

UI_ATTACHMENT(UI_WINDOW_OBJECT *_object,
              ATCF_FLAGS _atcFlags = ATCF_NO_FLAGS, int _offset = 0);
or
UI_ATTACHMENT(UI_WINDOW_OBJECT *_object,
              UI_WINDOW_OBJECT *_reference, ATCF_FLAGS _atcFlags, int _offset = 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 UI\_ATTACHMENT class object.

The first overloaded constructor creates a new UI\_ATTACHMENT object that ties the managed object to its parent.

- *\_object<sub>in</sub>* is the object to be managed.

- *\_atcFlags<sub>in</sub>* are flags that define the operation of the UI\_ATTACHMENT class. The following flags (declared in **UI\_WIN.HPP**) control the general operation of a UI\_ATTACHMENT class object:

**ATCF\_BOTTOM**—Maintains the bottom edge of the managed object at the specified distance from the object to which it is tied.

**ATCF\_LEFT**—Maintains the left edge of the managed object at the specified distance from the object to which it is tied.

**ATCF\_OPPOSITE**—Causes the managed object to be tied to the opposite edge of the object to which it is tied. For example, if the ATCF\_TOP flag is set, the top edge of the managed object will be tied to the bottom edge of the object to which it is tied.

**ATCF\_NO\_FLAGS**—Does not associate any special flags with the UI\_ATTACHMENT class. This flag should not be used in conjunction with any other ATCF flags.

**ATCF\_RIGHT**—Maintains the right edge of the managed object at the specified distance from the object to which it is tied.

**ATCF\_STRETCH**—Causes the managed object to be stretched, if necessary, to maintain its attachments. For example, if the left and right edges of the object are tied to the parent window and the window is sized, the managed object must stretch or shrink to maintain its distance from the edges.

**ATCF\_TOP**—Maintains the top edge of the managed object at the specified distance from the object to which it is tied.

- *\_offset<sub>in</sub>* is how far the managed object should be positioned from the object to which it is tied. This value is specified in cell dimensions.

The second overloaded constructor creates a new UI\_ATTACHMENT object that ties the managed object to a sibling object.

- *\_object<sub>in</sub>* is the object to be managed.
- *\_reference<sub>in</sub>* is the sibling object to which the *object* should be tied.
- *\_atcFlags<sub>in</sub>* are flags that define the operation of the UI\_ATTACHMENT class. For more details, see the description of *\_atcFlags* with the first constructor.



- `_offset_in` is how far the managed object should be positioned from the object to which it is tied. This value is specified in cell dimensions.

## **UI\_ATTACHMENT::~~UI\_ATTACHMENT**

### **Syntax**

```
#include <ui_win.hpp>

virtual ~UI_ATTACHMENT(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 UI\_ATTACHMENT object.

## **UI\_ATTACHMENT::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<sub>out</sub>* 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<sub>in</sub>* is a request to get or set information associated with the object. The following requests (defined in **UI\_WIN.HPP**) are recognized by the attachment 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 **ATCF\_FLAGS** that contains the flags to be cleared. 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 **ATCF\_FLAGS**.

**I\_GET\_OFFSET**—Returns the *offset* associated with the attachment. If this message is sent, *data* must be a pointer to a variable of type **int** where the attachment's offset will be copied.

**I\_GET\_REFERENCE\_OBJECT**—Returns the *reference* object associated with the attachment. If this message is sent, *data* should be a doubly indirected pointer to **UI\_WINDOW\_OBJECT**. *data* will be set to point to *reference*. If *data* is NULL, a pointer to *reference* is returned.

**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 **ATCF\_FLAGS** that contains the flags to be set. This request only sets those flags that are passed in; it does not clear any flags that are already set.

**I\_SET\_OFFSET**—Sets the *offset* associated with the attachment. If this message is sent, *data* must be a pointer to a variable of type **int** that contains the attachment's new offset.

**I\_SET\_REFERENCE\_OBJECT**—Sets the *reference* object associated with the attachment. If this message is sent, *data* must be a pointer to the sibling object that the managed object should be tied to.

All other requests are passed to `UI_CONSTRAINT::Information( )` for processing.

- `datairi/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.
- `objectIDin` is not used.

## UI\_ATTACHMENT::Modify

### Syntax

```
#include <ui_win.hpp>

virtual void Modify(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 updates the managed object's position and, if necessary, size according to the constraints specified for the attachment. The geometry manager calls each constraint's `Modify( )` function whenever the parent object's position is changed.

## Storage Members

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

## UI\_ATTACHMENT::UI\_ATTACHMENT

### Syntax

```
#include <ui_win.hpp>

UI_ATTACHMENT(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 UI\_ATTACHMENT by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a constraint is stored in a data file it is usually stored as part of a geometry manager and will be loaded when the geometry manager is loaded.

- *name<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* 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."
- *object<sub>in</sub>* 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."
- *objectTable<sub>in</sub>* 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 this manual. 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<sub>in</sub>* 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 this manual. 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.

## **UI\_ATTACHMENT::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 UI\_ATTACHMENT from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* 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."

- *object<sub>in</sub>* 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."
- *objectTable<sub>in</sub>* 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 this manual. 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<sub>in</sub>* 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 this manual. 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.

## UI\_ATTACHMENT::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<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* 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."
- *object<sub>in</sub>* 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."
- *objectTable<sub>in</sub>* 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 this manual. 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<sub>in</sub>* 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 this manual. 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.

## UI\_ATTACHMENT::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<sub>out</sub>* is a pointer to the object's **New( )** function.

## UI\_ATTACHMENT::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<sub>in</sub>* is the name of the object to be stored.
- *file<sub>in</sub>* 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."



*object<sub>in</sub>* 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`."

*objectTable<sub>in</sub>* 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 this manual. 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<sub>in</sub>* 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 this manual. 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-UI\_BGI\_DISPLAY

The UI\_BGI\_DISPLAY class implements a graphics display that uses the Borland BGI graphics library package to draw to the screen. Since the UI\_BGI\_DISPLAY class is derived from UI\_DISPLAY, only details specific to the UI\_BGI\_DISPLAY class are given in this chapter. For descriptions and examples regarding virtual or inherited display members, see "Chapter 7—UI\_DISPLAY."

The UI\_BGI\_DISPLAY class is declared in **UI\_DSP.HPP**. Its public and protected members are:

```
Class ZIL_EXPORT CLASS UI_BGI_DISPLAY : public UI_DISPLAY,
public UI_REGION_LIST
{
public:
    struct BGIFONT
    {
        int font;
        int charSize;
        int multX, divX;
        int multY, divY;
        int maxWidth, maxHeight;
    };
    typedef char BGIPATTERN[8];

    static UI_PATH *searchPath;
    static BGIFONT fontTable[ZIL_MAXFONTS];
    static BGIPATTERN patternTable[ZIL_MAXPATTERNS];

    UI_BGI_DISPLAY(int driver = 0, int mode = 0);
    virtual ~UI_BGI_DISPLAY(void);
    virtual void Bitmap(ZIL_SCREENID screenID, int column, int line,
        int bitmapWidth, int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette = ZIL_NULLP(UI_PALETTE),
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
        ZIL_BITMAP_HANDLE *ColorBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE),
        ZIL_BITMAP_HANDLE *monoBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE));
    virtual void BitmapArrayToHandle(ZIL_SCREENID screenID, int bitmapWidth,
        int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette, ZIL_BITMAP_HANDLE *ColorBitmap,
        ZIL_BITMAP_HANDLE *monoBitmap);
    virtual void BitmapHandleToArray(ZIL_SCREENID screenID,
        ZIL_BITMAP_HANDLE colorBitmap, ZIL_BITMAP_HANDLE monoBitmap,
        int *bitmapWidth, int *bitmapHeight, ZIL_UINT8 **bitmapArray);
    virtual void Ellipse(ZIL_SCREENID screenID, int column, int line,
        int startAngle, int endAngle, int xRadius, int yRadius,
        const UI_PALETTE *palette, int fill = FALSE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void IconArrayToHandle(ZIL_SCREENID screenID, int iconWidth,
        int iconHeight, const ZIL_UINT8 *iconArray,
        const UI_PALETTE *palette, ZIL_ICON_HANDLE *icon);
    virtual void IconHandleToArray(ZIL_SCREENID screenID,
        ZIL_ICON_HANDLE icon, int *iconWidth, int *iconHeight,
        ZIL_UINT8 **iconArray);
    virtual void Line(ZIL_SCREENID screenID, int column1, int line1,
        int column2, int line2, const UI_PALETTE *palette, int width = 1,
        int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual ZIL_COLOR MapColor(const UI_PALETTE *palette, int isForeground);
    virtual void Polygon(ZIL_SCREENID screenID, int numPoints,
```

```

        const int *polygonPoints, const UI_PALETTE *palette,
        int fill = FALSE, int _xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void Rectangle(ZIL_SCREENID screenID, int left, int top,
        int right, int bottom, const UI_PALETTE *palette, int width = 1,
        int fill = FALSE, int _xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void RectangleXORDiff(const UI_REGION &oldRegion,
        const UI_REGION &newRegion, ZIL_SCREENID screenID = ID_SCREEN,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void RegionDefine(ZIL_SCREENID screenID, int left, int top,
        int right, int bottom);
virtual void RegionMove(const UI_REGION &oldRegion, int newColumn,
        int newLine, ZIL_SCREENID oldScreenID = ID_SCREEN,
        ZIL_SCREENID newScreenID = ID_SCREEN);
virtual void Text(ZIL_SCREENID screenID, int left, int top,
        const ZIL_ICHAR *text, const UI_PALETTE *palette, int length = -1,
        int fill = TRUE, int _xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
        ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextHeight(const ZIL_ICHAR *string,
        ZIL_SCREENID screenID = ID_SCREEN,
        ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextWidth(const ZIL_ICHAR *string,
        ZIL_SCREENID screenID = ID_SCREEN,
        ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int VirtualGet(ZIL_SCREENID screenID, int left, int top,
        int right, int bottom);
virtual int VirtualPut(ZIL_SCREENID screenID);

protected:
    int maxColors;
    signed char virtualCount;
    UI_REGION virtualRegion;
    char _stopDevice;

    void SetFont(ZIL_LOGICAL_FONT logicalFont);
    void SetPattern(const UI_PALETTE *palette, int _xor);
};

```

## General Members

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

- *BGIFONT* is a structure that contains the following font information:

*font* contains the value of the font. `FNT_SMALL_FONT` (font is 0), `FNT_DIALOG_FONT` (font is 1) and `FNT_SYSTEM_FONT` (font is 2) are pre-defined by OpenZinc.

*charSize* can be used to magnify the size of a character. For more information see `settextstyle( )` in the *Borland C++ Library Reference*.

*multX*, *divX*, *multY* and *divY* provide additional ways to scale the font. For more information see `setusercharsize()` in the *Borland C++ Library Reference*.

*maxHeight* is the height of the tallest character.

*maxWidth* is the width of the widest character.

- *BGIPATTERN* is an array of 8 bytes that make up the 8x8 bitmap pattern. Each byte (8 bits) corresponds to 8 pixels in the pattern. The patterns defined by OpenZinc are: `PTN_SOLID_FILL`, `PTN_INTERLEAVE_FILL` and `PTN_BACKGROUND_FILL`. For more information see `setfillpattern()` in the *Borland C++ Library Reference*.
- *searchPath* contains the path to be searched for the BGI drivers or CHR font files. The BGI graphics library must have access to the BGI drivers in order to initialize graphics mode. Unless the BGI drivers are linked in, they must be accessible at run-time. Similarly, an application may rely on CHR font definition files. These files provide font information and, if used, must be accessible at run-time. The default fonts used by OpenZinc are linked into the application and thus do not require CHR files. Setting *searchPath* to include a path node for the BGI or CHR files will allow the application to run properly in graphics mode.
- *fontTable* is an array of *BGIFONT*. The default array contains space for 10 *BGIFONT* entries. The following entries are pre-defined by OpenZinc:

**FNT\_SMALL\_FONT**—A font used to display an icon's text string.

**FNT\_DIALOG\_FONT**—A font used when text is displayed on window objects (e.g., `UIW_BUTTON`, `UIW_STRING`, `UIW_TEXT`, etc.).

**FNT\_SYSTEM\_FONT**—A sans-serif style font used to display a window's title.

**NOTE:** When these three fonts are used, no CHR files are needed since they are linked into OpenZinc Application Framework. However, if other "stroked" fonts are added to this table, the proper CHR files must either be in the current path or be linked into the application.

The remaining entries in *fontTable* are initially set to Borland's `DEFAULT_FONT`, a fixed-width, 8x8, bitmapped font.

See the description of the *UI\_WINDOW\_OBJECT::font* member variable in "Chapter 43—`UI_WINDOW_OBJECT`" for information on specifying which font an object uses.

- *patternTable* is an array of *BGIPATTERN*. The default array contains space for 15 *BGIPATTERN* entries. The following entries are pre-defined by OpenZinc:

**PTN\_SOLID\_FILL**—A solid fill pattern.

**PTN\_INTERLEAVE\_FILL**—An interleaving line fill pattern.

**PTN\_BACKGROUND\_FILL**—The background fill pattern.

- *maxColors* is the maximum number of colors supported by the graphics mode that was initialized. For example, an EGA display might support sixteen colors. This member will be filled in according to information obtained from the BGI graphics library after it has initialized. The BGI graphics library has limited support for SVGA modes, including 256 color mode. OpenZinc will support whatever mode is initialized by the BGI graphics library.
- *\_virtualCount* is a count of the number of virtual screen operations that have taken place. For example, when the **VirtualGet()** function is called, *\_virtualCount* is decremented. Additionally, when the **VirtualPut()** function is called, *\_virtualCount* is incremented.
- *\_virtualRegion* is the region affected by either **VirtualGet( )** or **VirtualPut( )**.
- *\_stopDevice* is a variable used to prevent recursive updates of device images on the display. If *\_stopDevice* is TRUE, no drawing will be done to the screen. Otherwise, drawing will be made directly to the screen display.

## UI\_BGI\_DISPLAY::UI\_BGI\_DISPLAY

### Syntax

```
#include <ui_dsp.hpp>
```

```
UI_BGI_DISPLAY(int driver = 0, int mode = 0);
```

### 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 `UI_BGI_DISPLAY` object. When a new `UI_BGI_DISPLAY` class is constructed, the system finds the associated BGI device driver and sets the screen display to the background color and pattern specified by the inherited variable `backgroundPalette`.

- `driverin` and `modein` are arguments passed to the Borland `initgrah()` function. The argument `driver` specifies the graphics driver to be used. (The value 0 indicates an auto-detection.) The argument `mode` specifies the initial graphics mode (used only if `driver` is not auto-detect). For more information on these arguments see `initgraph()` in the *Borland C++ Library Reference* manual.

## Example 1

```
#include <ui_win.hpp>

main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_BGI_DISPLAY;

    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;

    UI_WINDOW_MANAGER *windowManager =
        new UI_WINDOW_MANAGER(display, eventManager);

    // Restore the system,
    delete windowManager;
    delete eventManager;
    delete display;
    return (0);
}
```

## Example 2

This example shows how a different font can be installed into the `fontTable` so that it may be used by the system.

```
#include <ui_win.hpp>

main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_BGI_DISPLAY;
    if (display->installed)
```

```

{
    // Set up a BGIFONT structure with the Borland default font.
    UI_BGI_DISPLAY::BGIFONT BGIFont = { DEFAULT_FONT, 1, 1, 1, 1, 1, 8, 8 };
    UI_BGI_DISPLAY::fontTable[5] = BGIFont;

    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;

    UI_WINDOW_MANAGER *windowManager =
        new UI_WINDOW_MANAGER(display, eventManager);

    // Restore the system,
    delete windowManager;
    delete eventManager;
    delete display;
    return (0);
}

```

## UI\_BGI\_DISPLAY::~~UI\_BGI\_DISPLAY

### Syntax

```

#include <ui_dsp.hpp>

~UI_BGI_DISPLAY(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 UI\_BGI\_DISPLAY class. Care should be taken to only destroy a UI\_BGI\_DISPLAY class that is not attached to another associated object.

## UI\_BGI\_DISPLAY::SetFont

### Syntax

```
#include <ui_dsp.hpp>

void SetFont(ZIL_LOGICAL_FONT logicalFont);
```

### 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 font information used by the BGI graphics library. The information contained in the *logicalFont* entry of the *fontTable* array is used to set the font.

- *logicalFont<sub>m</sub>* is the font to be used. *logicalFont* is an entry into the *fontTable* array.

## UI\_BGI\_DISPLAY::SetPattern

### Syntax

```
#include <ui_dsp.hpp>

void SetPattern(const UI_PALETTE *palette, int __xor)\
```

### 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 pattern information used by the BGI graphics library. The information contained in *palette* is used to set the pattern.

- *palette<sub>in</sub>* contains the pattern style, foreground color, and background color to be used when setting the pattern.
- *\_xor<sub>in</sub>* indicates if the pattern should be drawn with the xor attribute on. If *\_xor* is TRUE, the pattern will be an xor pattern. Otherwise, the pattern will not be xor.

## CHAPTER 4-UI\_CONSTRAINT

The UI\_CONSTRAINT class is the base class for all geometry management constraint classes used in OpenZinc. A constraint defines where an object can be positioned or how it can be sized in relation to its parent or sibling objects. Constraints are added to a geometry manager. The UI\_CONSTRAINT class is an abstract class that defines the functionality that must exist in each derived constraint class. Only derived constraint classes, such as UI\_ATTACHMENT, UI\_DIMENSION\_CONSTRAINT and UI\_RELATIVE\_CONSTRAINT, can be created.

The UI\_CONSTRAINT class is declared in **UI\_WIN.HPP**. Its public and protected members are:

```
class UI_CONSTRAINT : public UI_ELEMENT
{
public:
    UI_CONSTRAINT(UI_WINDOW_OBJECT *_object);
    virtual ~UI_CONSTRAINT(void);

    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);
    virtual void Modify(void) = 0;
    UI_GEOMETRY_MANAGER *Manager(UI_GEOMETRY_MANAGER *_manager =
        ZIL_NULLP(UI_GEOMETRY_MANAGER));
    ZIL_OBJECTID SearchID(void);

    // List members.
    UI_CONSTRAINT *Next(void);
    UI_CONSTRAINT *Previous(void);

#ifdef ZIL_LOAD
    static UI_CONSTRAINT *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));
    UI_CONSTRAINT(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
#ifdef 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 *object;
    ZIL_NUMBERID numberID;
    ZIL_OBJECTID searchID;
    UI_GEOMETRY_MANAGER *manager;
};
```

## General Members

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

- *object* is the object being managed by the constraint.
- *numberID* is the numberID that identifies this constraint uniquely in the geometry manager's list.
- *searchID* is an objectID that identifies what type of constraint this is.
- *manager* is a pointer to the geometry manager with which this constraint is associated.

## UI CONSTRAINT::UI CONSTRAINT

### Syntax

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

```
UI_CONSTRAINT(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 constructor creates a new UI\_CONSTRAINT object.

- *\_object<sub>in</sub>* is the object to be managed by the constraint.

## UI\_CONSTRAINT::~~UI\_CONSTRAINT

### Syntax

```
#include <ui_win.hpp>

virtual ~UI_CONSTRAINT(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 UI\_CONSTRAINT object.

## UI\_CONSTRAINT::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<sub>out</sub>* 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<sub>in</sub>* is a request to get or set information associated with the object. The following requests (defined in **UI\_WIN.HPP**) are recognized by the constraint object:

**I\_GET\_NUMBERID**—Returns the numberID of this constraint. If this request is sent, *data* should be a pointer to ZIL\_NUMBERID.

**I\_GET\_OBJECT**—Returns the *object* associated with the constraint. If this message is sent, *data* should be a doubly indirected pointer to UI\_WINDOW\_OBJECT. *data* will be set to point to *object*. If *data* is NULL, a pointer to *object* is returned.

**I\_SET\_NUMBERID**—Sets the numberID of this constraint. If this request is sent, *data* should be a pointer to ZIL\_NUMBERID.

**I\_SET\_OBJECT**—Sets the *object* associated with the constraint. If this message is sent, *data* must be a pointer to the object to be managed.

- *data<sub>in/out</sub>* 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<sub>in</sub>* is not used.

## UI\_CONSTRAINT::Manager

### Syntax

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

```
UI_GEOMETRY_MANAGER *Manager(UI_GEOMETRY_MANAGER *_manager =  
    ZIL_NULLP(UI_GEOMETRY_MANAGER));
```

## 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 either set the *manager* member or get a pointer to it.

- *returnValue<sub>out</sub>* is a pointer to *manager*, the geometry manager with which this constraint is associated.
- *\_manager<sub>in</sub>* is a pointer to the geometry manager with which this constraint is associated. If *\_manager* is NULL, the *manager* member will not be modified, but a pointer to *manager* will be returned.

## UI\_CONSTRAINT::Modify

### Syntax

```
#include <ui_win.hpp>

virtual void Modify(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 updates the managed object's position and, if necessary, size according to the constraints specified. The geometry manager calls each constraint's **Modify**( ) function whenever the parent object's position is changed.

## **UI\_CONSTRAINT::Next**

### **Syntax**

```
#include <ui_win.hpp>

UI_CONSTRAINT*Next(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 next constraint, if one exists, in the list of constraints.

- *returnValue<sub>out</sub>* is a pointer to the next constraint in the list. If there is not a next constraint, *returnValue* is NULL.

## **UI\_CONSTRAINT::Previous**

### **Syntax**

```
#include <ui_win.hpp>

UI_CONSTRAINT*Previous(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 previous constraint, if one exists, in the list of constraints.

- *returnValue<sub>out</sub>* is a pointer to the previous constraint in the list. If there is not a previous constraint, *returnValue* is NULL.

## UI\_CONSTRAINT::SearchID

### Syntax

```
#include <ui_win.hpp>

ZIL_OBJECTID SearchID(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 constraint's *searchID*.

## Storage Members

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



## UI\_CONSTRAINT::UI\_CONSTRAINT

### Syntax

```
#include <ui_win.hpp>

UI_CONSTRAINT(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 `UI_CONSTRAINT` by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a constraint is stored in a data file it is usually stored as part of a geometry manager and will be loaded when the geometry manager is loaded.

- *name<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* 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`."
- *object<sub>in</sub>* 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`."
- *objectTable<sub>in</sub>* 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 this manual. 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<sub>in</sub>* 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 this manual. 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.

## **ULCONSTRAINT::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 UI\_CONSTRAINT from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* 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."
- *object<sub>in</sub>* 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."

- *objectTable<sub>m</sub>* 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 this manual. 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<sub>m</sub>* 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 this manual. 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.

## ULCONSTRAINT::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<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* 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."
- *object<sub>in</sub>* 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."
- *objectTable<sub>in</sub>* 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 this manual. 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<sub>in</sub>* 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 this manual. 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.

## UI\_CONSTRAINT::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*<sub>out</sub> is a pointer to the object's **New()** function.

## UI\_CONSTRAINT::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.

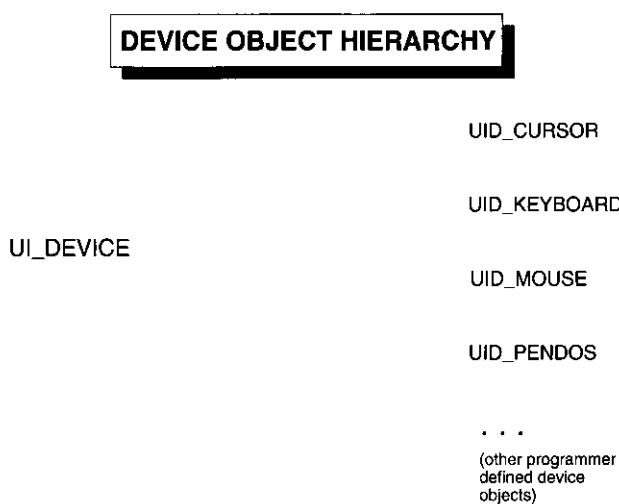
- *name<sub>in</sub>* is the name of the object to be stored.
- *file<sub>in</sub>* is a pointer to the ZIL\_STORAGE where the persistent object will be stored. For more information on persistent object files, see "Chapter 66—ZIL\_STORAGE."

- *object<sub>in</sub>* 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."
- *objectTable<sub>in</sub>* 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 this manual. 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<sub>in</sub>* 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 this manual. 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-UI\_DEVICE

The UI\_DEVICE class is an abstract class that defines basic information associated with input devices (e.g., keyboard, mouse). Since the UI\_DEVICE class is abstract, it cannot be used as a constructed class. Rather, derived classes, such as UID\_KEYBOARD, UID\_CURSOR, or UID\_MOUSE must be used. The figure below shows the device object hierarchy:



Classes derived from the UI\_DEVICE base class include:

**UID\_CURSOR**—A blinking cursor shown on the screen. In text mode, this device is implemented as the hardware cursor. In graphics mode, this device paints a blinking cursor on the screen.

**UID\_KEYBOARD**—A polled keyboard interface that retrieves keyboard information from the end-user.

**UID\_MOUSE**—A polled mouse device that receives mouse information from the end-user.

**UID\_TIMER**—A timer device that generates timer messages when a period of time specified by the programmer has expired.



**Other programmer defined device objects**—Any other programmer defined device that conforms to the operating protocol defined by the `UI_DEVICE` base class.

Input devices are attached to the Event Manager at run-time by the programmer. Once a device is attached, it feeds input information to the event queue when polled by the Event Manager, or it feeds directly to the event queue if it is an interrupt device.

The `UI_DEVICE` class is declared in `UI_EVT.HPP`. Its public and protected members are:

```
enum ALT_STATE
{
    ALT_NOT_PRESSED,
    ALT_PRESSED_NO_EVENTS,
    ALT_PRESSED_EVENTS
};

class ZIL_EXPORT_CLASS UI_DEVICE : public UI_ELEMENT
public:
    static ALT_STATE altState;
    static UI_DISPLAY *display;
    static UI_EVENT_MANAGER *eventManager;

    int installed;
    ZIL_DEVICE_TYPE type;
    ZIL_DEVICE_STATE state;

    virtual ~UI_DEVICE(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event) = 0;

    // List members.
    UI_DEVICE *Next(void);
    UI_DEVICE *Previous(void);

protected:
    UI_DEVICE(ZIL_DEVICE_TYPE type, ZIL_DEVICE_STATE State);
    static int CompareDevices(void *device1, void *device2);
    virtual void Poll(void) = 0;
```

## General Members

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

- `ALT_STATE` contains values that are used to indicate the status of the <ALT> key when an event occurs. The following values are used:

**ALT\_NOT\_PRESSED**—The <ALT> key has not been pressed.

**ALT\_PRESSED\_NO\_EVENTS**—The <ALT> key has been pressed, but no other input information has been received since the key was pressed.

**ALT\_PRESSED\_EVENTS**—The <ALT> key continues to be pressed while another event has been received.

- *altState* is a static variable that indicates whether the keyboard <ALT> key is being pressed. It is used by the keyboard and mouse to detect the selection of <ALT> keys or else to send an <ALT> message if the <ALT> key is pressed and then released with no other keyboard or mouse event information.
- *display* is a pointer to a constructed display class. This variable is automatically set when the derived device is added to the Event Manager.
- *eventManager* is a pointer to a constructed Event Manager class. This variable is automatically set when the derived device is added to the Event Manager.
- *installed* indicates whether the input device was able to initialize itself. If installation is successful, *installed* is TRUE. If installation is not successful—for instance, if the mouse input device cannot find a mouse driver—*installed* is FALSE.
- *type* is the type of device that has been created. For example, the keyboard generates a type of E\_KEY, the mouse generates a type of E\_MOUSE and the cursor generates a type of E\_CURSOR. Every device created either has a unique type or else it must have the generic device type E\_DEVICE. The Event Manager uses the *type* information to route messages it receives to the proper device. The Event Manager receives device events via its **Event()** member function.
- *state* describes the current state of the device. For example, the device may be on or off. The state can be any of several D\_ codes, defined in **UI\_EVT.HPP**. These codes are: D\_OFF, D\_ON, and D\_HIDE. For example, if a keyboard device is in the D\_OFF state, it will not generate any events. If a mouse is in the D\_HIDE state, it will still be active, but it will not be visible on the screen.

## UI\_DEVICE::UI\_DEVICE

### Syntax

```
#include <ui_evt.hpp>
```

```
UI_DEVICE(ZIL_DEVICE_TYPE type, ZIL_DEVICE_STATE state);
```

## 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 initializes the information associated with all devices derived from the `UI_DEVICE` base class. The `UI_DEVICE` constructor is protected since `UI_DEVICE` is an abstract class (i.e., only derived instances of `UI_DEVICE` can be made.)

- *type<sub>in</sub>* specifies the type of derived device that is to be initialized. The contents of this argument are copied into the protected *type* member variable. OpenZinc Application Framework reserves the values 0 through 99 for raw input devices. The following devices are defined within OpenZinc Application Framework:

**E\_CURSOR(50)**—Identification for the `UID_CURSOR` class.

**E\_DEVICE(99)**—Identification used to define a generic device.

**E\_KEY(10)**—Identification for the `UID_KEYBOARD` class.

**E\_MACINTOSH(4)**—Identification for Macintosh events.

**E\_MOTIF(3)**—Identification for Motif events.

**E\_MOUSE(30)**—Identification for the `UID_MOUSE` class.

**E\_MSWINDOWS(1)**—Identification for **MS** Windows events.

**E\_NEXTSTEP(11)**—Identification for NEXTSTEP events.

**E\_OS2(2)**—Identification for OS/2 events.

The *type* value associated with each device is significant, because the Event Manager polls devices in ascending order. Each derived device class must either use the `E_DEVICE` type or have a type that is unique and is within the 0-99 value restrictions. The following additional raw device identifications are reserved by OpenZinc

Application Framework for future use: 12-19, 31-39, 51-59, 70-79 and 90-98. The remaining values 5-9, 20-29, 40-49, 60-69 and 80-89 can be used by the programmer.

- *initialState<sub>m</sub>* specifies the initial state of the derived device. The information contained in this argument depends on the type of device created but should either be D\_OFF or D\_ON.

## Example

```
#include <ui_evt.hpp>

UID_CURSOR::UID_CURSOR(ZIL_DEVICE_STATE initialState) :
    UI_DEVICE(E_CURSOR, initialState)
{
}
}
```

## UI\_DEVICE::~~UI\_DEVICE

### Syntax

```
#include <ui_evt.hpp>

virtual ~UI_DEVICE(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 UI\_DEVICE object. It is called when a derived device class is destroyed.

## UI\_DEVICE::CompareDevices

### Syntax

```
#include <ui_evt.hpp>

static int CompareDevices(void *device1, void *device2)\
```

### 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 compare two devices according to their types.

- *returnValue<sub>out</sub>* is 0 if the two devices have the same type. *returnValue* is positive if *device1* has a greater device type than *device2*. *returnValue* is negative if *device1* has a lower device type than *device2*.
- *device1<sub>in</sub>* is a pointer to a UI\_DEVICE object.
- *device2<sub>in</sub>* is a pointer to a UI\_DEVICE object.

### Example

```
#include <ui_evt.hpp>

UI_EVENT_MANAGER::UI_EVENT_MANAGER(UI_DISPLAY *_display, int _noOfElements) :
    UI_LIST(UI_DEVICE::CompareDevices), queueBlock(_noOfElements), level(1)
{
    display = _display;
    UI_DEVICE::display = display;
    UI_DEVICE::eventManager = this;
}
```

## UI\_DEVICE::Event

### Syntax

```
#include <ui_evt.hpp>

virtual EVENT_TYPE Event(const UI_EVENT &event) = 0;
```

### 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 a pure virtual function, so it has no declaration. This means that every class which is derived from UI\_DEVICE must have an **Event()** function.

The **Event()** function is used to communicate run-time state information to a particular device. For example, the **UID\_KEYBOARD::Event()** function can receive event information to turn on or off keyboard input. The type of information required by the **Event()** function depends on the type of device that is to receive the message.

The *event.type* value contains the state information to be passed to the device. For example, if a programmer wanted to send the D\_OFF message to all timer devices in the Event Manager, the following code could be used:

```
event.type = D_OFF;
event.region.left = 0;           // Define the whole screen,
event.region.top = 0;
event.region.right = display->columns - 1;
event.region.bottom = display->lines - 1;
eventManager->Event(event, E_TIMER);
```

The following general messages can be sent to a device:

**D\_HIDE**—Hides the device while the application paints information to the screen. This advanced message prevents the device from leaving blank areas on the screen when low-level screen painting operations are performed. In general, a programmer should not use this message. Window objects and display classes automatically hide the input devices when they paint information to the screen. If the D\_HIDE message is used, the *event.region* information must contain the region that will be re-painted.

This allows the affected device to only turn itself off when the affected region overlaps the device's screen position. This message must be used in conjunction with the `D_ON` message (described below) and should be used in the following order:

```
// Hide all devices before painting information to the screen.
UI_EVENT event;
event.type = D_HIDE;
event.region.left = 0;           // Define the whole screen,
event.region.top = 0;
event.region.right = display->columns - 1;
event.region.bottom = display->lines - 1;
eventManager->Event(event, E_DEVICE);
// Paint information directly to the screen.

// Show all devices that may have been shut off.
event.type = D_ON;              // event.region was defined previously
eventManager->Event(event, E_DEVICE);
```

**D\_OFF**—Turns the device off.

**D\_ON**—Turns the device on.

**S\_DEINITIALIZE**—Serves as a warning that the device is being subtracted from the Event Manager. This allows the device to halt further execution and/or prepare to be deleted.

**S\_INITIALIZE**—Initializes internal information associated with the device. This message can be used when the device cannot initialize all its information at the time that the class constructor is called.

**S\_POSITION**—Changes the screen position of the device. If this message is sent, *event.position.column* and *event.position.line* must contain the run-time display position of the device on the screen. The values of *event.position.column* and *event.position.line* depend on the type of display mode in which the application is running. For example, if a `UID_CURSOR` object is to be positioned at the center of the screen while the application is running in text mode (e.g., an 80 column by 25 line screen) the position values should be:

```
event.position.column = 40;
event.position.line = 13;
```

If, on the other hand, the application is running in a 640 column by 480 line graphics mode, the position values should be:

```
event.position.column = 320;
event.position.line = 240;
```

If the device is in a D\_OFF state, the position change should be reflected when the device is turned back on.

In addition to the messages described above, each device may be sent private messages defined by the programmer. OpenZinc Application Framework reserves message numbers 0x0000 through 0x00FF. (Programmers may use any unsigned values greater than 0x00FF.)

**NOTE:** The chapters for the UID\_CURSOR, UID\_KEYBOARD and UID\_MOUSE classes contain more information about some messages specific to those classes.

## Example

```
#include <ui_evt.hpp>

main()
{
    // Attach the keyboard to the event manager.
    UI_TEXT_DISPLAY *display = new UI_TEXT_DISPLAY;
    UI_EVENT_MANAGER eventManager(display);
    eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;

    // Turn all the devices off.
    UI_EVENT event;
    event.type = D_OFF;
    for (UI_DEVICE *device = eventManager.First(); device;
         device = device->Next())
        device->Event(event);
}
```

## UI\_DEVICE::Next

### Syntax

```
#include <ui_win.hpp>

UI_DEVICE *Next(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 next device, if one exists, in the list of devices.

## Example

```
UI_DISPLAY *display = new UI_TEXT_DISPLAY;
UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
*eventManager
+ new UID_KEYBOARD
+ new UID_MOUSE
+ new UID_CURSOR;

for (UI_DEVICE *device = eventManager->First(); device;
     device = device->Next());
```

## UI\_DEVICE::Poll

### Syntax

```
#include <ui_evt.hpp>

virtual void Poll(void) = 0;
```

## 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 a pure virtual function, so it has no declaration. This means that every class which is derived from `UI_DEVICE` must have a **Poll** function.

The **Poll()** function is used by `UI_EVENT_MANAGER::Get()` to give input devices time to put event information into the event queue. For example, in DOS the **UI\_KEYBOARD::Poll()** function gets information from the keyboard BIOS and enters that information, as `UI_EVENT` information, into the Event Manager's queue of events.

The **Poll()** function is called every time an event is requested from the Event Manager unless the `Q_NO_POLL` flag was set when requesting the event.

## Example

```
#include <ui_evt.hpp>

int UI_EVENT_MANAGER::Get(UI_EVENT kevent, Q_FLAGS flags)
{
    // Stay in loop while no event conditions are met.
    int error = -1;
    do
    {
        // Call all the polled devices.
        UI_DEVICE *device;
        if (IFlagSet (flags, Q_NO_POLL) )
            for (device = First(); device; device = device->Next())
                device->Poll ();

    } while (error);

    // Return the error status,
    return (error);
}
```

## UI\_DEVICE::Previous

### Syntax

```
#include <ui_win.hpp>

UI_DEVICE *Previous(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 previous device, if one exists, in the list of devices.

## Example

```
UI_DISPLAY *display = new UI_TEXT_DISPLAY;
UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
*eventManager
+ new UID_KEYBOARD
+ new UID_MOUSE
+ new UID_CURSOR;

for (UI_DEVICE *device = eventManager->Last(); device;
     device = device->Previous());
```

## CHAPTER 6 - UI\_DIMENSION\_CONSTRAINT

The UI\_DIMENSION\_CONSTRAINT class object is used for geometry management. Specifically, this class limits the width and height dimensions of a managed object. The UI\_DIMENSION\_CONSTRAINT is added to the parent object's geometry manager. See "Chapter 14—UI\_GEOMETRY\_MANAGER" for more details on using the geometry manager.

The UI\_DIMENSION\_CONSTRAINT class is declared in **UI\_WIN.HPP**. Its public and protected members are:

```
class UI_DIMENSION_CONSTRAINT : public UI_CONSTRAINT
{
public:
    UI_DIMENSION_CONSTRAINT(UI_WINDOW_OBJECT * object,
        DNCF_FLAGS_dncFlags = DNCF_NO_FLAGS, int _minimum = 0,
        int _maximum = 0);
    virtual ~UI_DIMENSION_CONSTRAINT(void);

    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);
    virtual void Modify(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));
    UI_DIMENSION_CONSTRAINT(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:
    DNCF_FLAGS dncFlags;
    int maximum;
    int minimum;
};
```

## General Members

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

- *dncFlags* are flags that define the operation of the `UI_DIMENSION_CONSTRAINT` class. A full description of the dimension constraint flags is given in the `UI_DIMENSION_CONSTRAINT` constructor.
- *maximum* is the maximum size allowed by the constraint. This value is specified in cell dimensions.
- *minimum* is the minimum size allowed by the constraint. This value is specified in cell dimensions.

## UI\_DIMENSION\_CONSTRAINT::UI\_DIMENSION\_CONSTRAINT

### Syntax

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

```
UI_DIMENSION_CONSTRAINT(UI_WINDOW_OBJECT *_object,  
    DNCF_FLAGS _dncFlags = DNCF_NO_FLAGS, int _minimum = 0,  
    int jmaximum = 0);
```

### 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 `UI_DIMENSION_CONSTRAINT` object.

- *\_object<sub>in</sub>* is the object to be managed.

- *\_dncFlags<sub>in</sub>* are flags that define the operation of the UI\_DIMENSION\_-CONSTRAINT class. The following flags (declared in **UI\_WIN.HPP**) control the general operation of a UI\_DIMENSION\_CONSTRAINT class object:

DNCF\_HEIGHT—Causes the constraint to manage the height of the managed object.

DNCF\_NO\_FLAGS—Does not associate any special flags with the UI\_DIMENSION\_CONSTRAINT class. This flag should not be used in conjunction with any other DNCF\_ flags.

DNCF\_WIDTH—Causes the constraint to manage the width of the managed object.

- *\_minimum<sub>in</sub>* is the minimum size allowed by the constraint. This value is specified in cell dimensions.
- *\_maximum<sub>in</sub>* is the maximum size allowed by the constraint. This value is specified in cell dimensions.

## UI\_DIMENSION\_CONSTRAINT::~~UI\_DIMENSION\_CONSTRAINT

### Syntax

```
#include <ui_win.hpp>

virtual ~UIJDIMENSION_CONSTRAINT(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 UI\_DIMENSION\_CONSTRAINT object.

## UI\_DIMENSION\_CONSTRAINT::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<sub>out</sub>* 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<sub>in</sub>* is a request to get or set information associated with the object. The following requests (defined in **UI\_WIN.HPP**) are recognized by the attachment 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 **DNCF\_FLAGS** that contains the flags to be cleared. 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 **DNCF\_FLAGS**. If *data* is NULL, a pointer to *dncFlags* will be returned.

**I\_GET\_MAXDIMENSION**—Returns the *maximum* size allowed by the constraint. If this message is sent, *data* must be a pointer to a variable of type **int** where the constraint's *maximum* will be copied.

**I\_GET\_MINDIMENSION**—Returns the *minimum* size allowed by the constraint. If this message is sent, *data* must be a pointer to a variable of type **int** where the constraint's *minimum* will be copied.

**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 **DNCF\_FLAGS** that contains the flags to be set. This request only sets those flags that are passed in; it does not clear any flags that are already set.

**I\_SET\_MAXDIMENSION**—Sets the *maximum* size allowed by the constraint. If this message is sent, *data* must be a pointer to a variable of type **int** that contains the constraint's new *maximum*.

**I\_SET\_MINDIMENSION**—Sets the *minimum* size allowed by the constraint. If this message is sent, *data* must be a pointer to a variable of type **int** that contains the constraint's new *minimum*.

All other requests are passed to **UI\_CONSTRAINT::Information()** for processing.

- *data<sub>in/out</sub>* 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<sub>in</sub>* is not used.

## **UI\_DIMENSION\_CONSTRAINT::Modify**

### **Syntax**

```
#include <ui_win.hpp>

virtual void Modify(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 ensures that the managed object's size is within the range specified for the constraint. The geometry manager calls each constraint's **Modify()** function whenever the parent object's position is changed.

## Storage Members

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

## UI\_DIMENSION\_CONSTRAINT::UI\_DIMENSION\_CONSTRAINT

### Syntax

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

```
UI_DIMENSION_CONSTRAINT(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 `UI_DIMENSION_CONSTRAINT` by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a constraint is stored in a data file it is usually stored as part of a geometry manager and will be loaded when the geometry manager is loaded.

- *name<sub>in</sub>* is the name of the object to be loaded.

- *file<sub>in</sub>* 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."
- *object<sub>in</sub>* 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."
- *objectTable<sub>in</sub>* 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 this manual. 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<sub>in</sub>* 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 this manual. 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.

## UI\_DIMENSION\_CONSTRAINT::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 `UI_DIMENSION_CONSTRAINT` from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- `namein` is the name of the object to be loaded.
- `filein` 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."
- `objectin` 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."
- `objectTablein` 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 this manual. 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.
- `userTablein` 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 this manual. 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.

## UI\_DIMENSION\_CONSTRAINT::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<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* 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."
- *object<sub>in</sub>* 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."
- *objectTable<sub>in</sub>* 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 *UIJWINDOWOBJECT::objectTable* in "Chapter 43—UIJWINDOW\_OBJECT" in this manual. 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<sub>in</sub>* 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 *UIJWINDOW\_OBJECT:-userTable* in "Chapter 43—UI\_WINDOW\_OBJECT" in this manual. 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.

## **UI\_DIMENSION\_CONSTRAINT::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<sub>in</sub>* is a pointer to the object's **New()** function.

## **UI\_DIMENSION\_CONSTRAINT::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

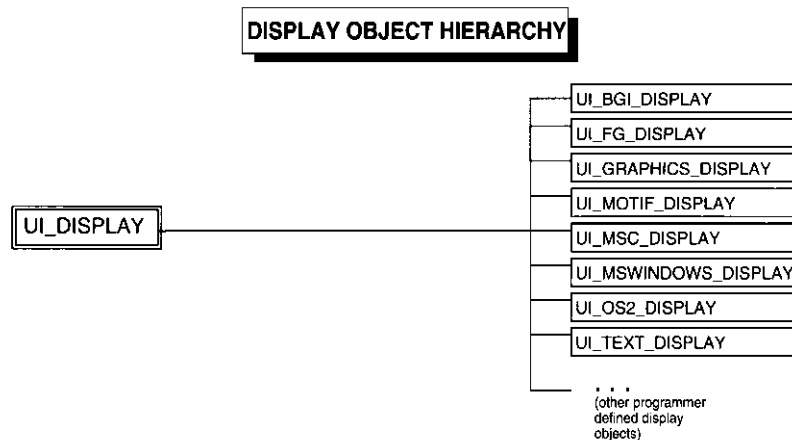
This advanced function is used to write an object to a data file.

- *name<sub>in</sub>* is the name of the object to be stored.
- *file<sub>in</sub>* 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."
- *object<sub>in</sub>* 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."
- *objectTable<sub>in</sub>* 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 this manual. 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<sub>m</sub>* 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 *UIWINDOW\_OBJECT::userTable* in "Chapter 43—UI\_WINDOW\_OBJECT" in this manual. 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-UI\_DISPLAY

The `UI_DISPLAY` class is the base class for all display classes used in OpenZinc. A display class is used to draw to the screen, whether in graphics mode, in text mode or in a graphical operating system. The `UI_DISPLAY` class defines the functionality that must exist in each derived display class. While the `UI_DISPLAY` class is not technically abstract (i.e., it contains no pure virtual functions), it should not be used as a constructed class. Rather, derived classes, such as `UI_BGI_DISPLAY`, `UI_MSWINDOWS_DISPLAY` or `UI_NEXTSTEP_DISPLAY` must be used. The figure below shows the display object hierarchy:



Classes derived from the `UI_DISPLAY` base class include:

**UI\_BGI\_DISPLAY**—A DOS graphics display that uses the Borland BGI graphics library to draw on the screen. The `UI_BGI_DISPLAY` class provides support for CGA, EGA, VGA and Hercules monochrome display adapters running in graphics mode.

**UI\_GRAPHICS\_DISPLAY**—A graphics display that uses the GFX graphics library to draw on the screen. The `UI_GRAPHICS_DISPLAY` class provides support for CGA, EGA, VGA, SVGA and Hercules monochrome display adapters running in graphics mode.

**UI\_MACINTOSH\_DISPLAY**—A graphics display class that uses the Macintosh graphics interface to draw within the Macintosh environment.



**UI\_MSC\_DISPLAY**—A DOS graphics display that uses the Microsoft C Graphics library to draw on the screen. The UI\_MSC\_DISPLAY class provides support for CGA, EGA, VGA, SVGA and Hercules monochrome display adapters running in graphics mode.

**UI\_MSWINDOWS\_DISPLAY**—A graphics display that uses the Microsoft Windows graphics interface to draw within the Windows environment.

**UI\_NEXTSTEP\_DISPLAY**—A graphics display class that uses the NEXTSTEP graphics interface to draw within the NEXTSTEP environment.

**UI\_OS2\_DISPLAY**—A graphics display class that uses the OS/2 graphics interface to draw within the OS/2 environment.

**UI\_TEXT\_DISPLAY**—A text display that draws to screen memory. The UI\_TEXT\_DISPLAY class provides support for MDA, CGA, EGA and VGA display adapters running DOS text mode or Curses. This includes the following modes of operation:

- 25 line x 40 column mode
- 25 line x 80 column mode
- 43 line x 80 column mode
- 50 line x 80 column mode

This class also contains support for snow checking (CGA monitors) and IBM TopView (which supports operation in the Microsoft Windows and Quarterdeck desqVIEW environments).

**UI\_WCC\_DISPLAY**—A DOS graphics display class that uses the Watcom graphics library to draw on the screen. The UI\_WCC\_DISPLAY class provides support for CGA, EGA, VGA, SVGA and Hercules monochrome display adapters running in graphics mode.

**UI\_XT\_DISPLAY**—A graphics display class that uses the Motif graphics interface to draw within the Motif environment.

**Other programmer defined screen display objects**—Any other programmer defined display class that conforms to the operating protocol defined by the UI\_DISPLAY base class.

By abstracting the display class, an application does not need to know which display class has been created. If, for example, the application needs to draw a rectangle, it can simply

call the display's virtual **Rectangle()** member function. The application can thus be written generically for all environments.

The `UI_DISPLAY` class is declared in `UI_DSP.HPP`. Its public and protected members are:

```

class ZIL_EXPORT_CLASS UI_DISPLAY : public ZIL_INTERNATIONAL
{
public:
    int installed;
    int isText;
    int isMono;
    int columns, lines;
    int cellWidth, cellHeight;
    int preSpace, postSpace;
    ZIL_INT32 miniNumeratorX, miniDenominatorX;
    ZIL_INT32 miniNumeratorY, miniDenominatorY;
    ZIL_ICHAR *operatingSystem;
    ZIL_ICHAR *windowingSystem;

    static UI_PALETTE *backgroundPalette;
    static UI_PALETTE *xorPalette;
    static UI_PALETTE *colorMap;

#ifdef ZIL_MSDOS || defined(ZIL_CURSES)
    static UI_PALETTE *markPalette;
#endif
#ifdef ZIL_MSWINDOWS
    #if defined(ZIL_MSWINDOWS)
        HANDLE hlnstance;
        HANDLE hPrevInstance;
        int nCmdShow;
    #endif
#endif
#ifdef ZIL_OS2
    HAB hab;
#endif
#ifdef ZIL_XT
    XtAppContext _appContext;
    Widget topShell;
    Display *xDisplay;
    Screen *xScreen;
    int xScreenNumber;
    GC xGc;
    GC xorGC;
    char *appClass;
    Pixmap interleaveStipple;
#endif

    virtual ~UI_DISPLAY(void);
    virtual void Bitmap(ZIL_SCREENID screenID, int column, int line,
        int bitmapWidth, int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette = ZIL_NULLP(UI_PALETTE),
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
        ZIL_BITMAP_HANDLE *colorBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE),
        ZIL_BITMAP_HANDLE *monoBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE));
    virtual void BitmapArrayToHandle(ZIL_SCREENID screenID, int bitmapWidth,
        int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette, ZIL_BITMAP_HANDLE *colorBitmap,
        ZIL_BITMAP_HANDLE *monoBitmap);
    virtual void BitmapHandleToArray(ZIL_SCREENID screenID,
        ZIL_BITMAP_HANDLE colorBitmap, ZIL_BITMAP_HANDLE monoBitmap,
        int *bitmapWidth, int *bitmapHeight, ZIL_UINT8 **bitmapArray);
    virtual void Ellipse(ZIL_SCREENID screenID, int column, int line,
        int startAngle, int endAngle, int xRadius, int yRadius,
        const UI_PALETTE *palette, int fill = FALSE, int _xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void IconArrayToHandle(ZIL_SCREENID screenID, int iconWidth,
        int iconHeight, const ZIL_UINT8 *iconArray,

```

```

    const UI_PALETTE *palette, ZIL_ICON_HANDLE *icon);
virtual void IconHandleToArray(ZIL_SCREENID screenID,
    ZIL_ICON_HANDLE icon, int *iconWidth, int *iconHeight,
    ZIL_UINT8 **iconArray);
virtual void Line(ZIL_SCREENID screenID, int column1, int line1,
    int column2, int line2, const UI_PALETTE *palette, int width = 1,
    int _xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual ZIL_COLOR MapColor(const UI_PALETTE *palette, int isForeground);
static RGBColor MapRGBColor(ZIL_COLOR fromColor);
virtual void Polygon(ZIL_SCREENID screenID, int numPoints,
    const int *polygonPoints, const UI_PALETTE *palette,
    int fill = FALSE, int _xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
void Rectangle(ZIL_SCREENID screenID, const UI_REGION &region,
    const UI_PALETTE *palette, int width = 1, int fill = FALSE,
    int _xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void Rectangle(ZIL_SCREENID screenID, int left, int top,
    int right, int bottom, const UI_PALETTE *palette, int width = 1,
    int fill = FALSE, int _xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void RectangleXORDiff(const UI_REGION &oldRegion,
    const UI_REGION &newRegion, ZIL_SCREENID screenID = ID_SCREEN,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
void RegionDefine(ZIL_SCREENID screenID, const UI_REGION &region);
virtual void RegionDefine(ZIL_SCREENID screenID, int left, int top,
    int right, int bottom);
virtual void RegionMove(const UI_REGION &oldRegion, int newColumn,
    int newLine, ZIL_SCREENID oldScreenID = ID_SCREEN,
    ZIL_SCREENID newScreenID = ID_SCREEN), -
virtual void Text(ZIL_SCREENID screenID, int left, int top,
    const ZIL_ICHAR *text, const UI_PALETTE *palette, int length = -1,
    int fill = TRUE, int _xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextHeight(const ZIL_ICHAR *string,
    ZIL_SCREENID screenID = ID_SCREEN,
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextWidth(const ZIL_ICHAR *string,
    ZIL_SCREENID screenID = ID_SCREEN,
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
int VirtualGet(ZIL_SCREENID screenID, const UI_REGION &region);
virtual int VirtualGet(ZIL_SCREENID screenID, int left, int top,
    int right, int bottom);
virtual int VirtualPut(ZIL_SCREENID screenID);

```

protected:

```

    struct ZIL_EXPORT_CLASS UI_DISPLAY_IMAGE
    {
        UI_REGION region;
        ZIL_UINT8 *image;
        ZIL_UINT8 *screen;
        ZIL_UINT8 *backup;
    };

    UI_DISPLAY_IMAGE displayImage[MAX_DISPLAY_IMAGES];

    UI_DISPLAY(int isText,
        const ZIL_ICHAR *operatingSystem = ZIL_NULLP(ZIL_ICHAR),
        const ZIL_ICHAR *windowingSystem = ZIL_NULLP(ZIL_ICHAR));
    int RegionInitialize(UI_REGION &region, const UI_REGION *clipRegion,
        int left, int top, int right, int bottom);

```

```

public:
    // Character encoding
    static ZIL_ICHAR codeSet[15];
};

```

## General Members

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

- *installed* indicates if the display class has been successfully installed, *installed* is set to FALSE by the UI\_DISPLAY class. Derived display classes will set this variable to TRUE if the display installs correctly.
- *isText* indicates whether the display is a text mode display or a graphics mode display. *isText* is TRUE if the application is running in text mode. Otherwise, *isText* is FALSE.
- *isMono* indicates if the display class is operating in monochrome mode. *isMono* is TRUE if the display is a monochrome display. Otherwise, it is FALSE.
- *columns* contains the number of columns that can be displayed in the current mode.
- *lines* contains the number of lines that can be displayed in the current mode.
- *cellWidth* is the width of a cell. OpenZinc logically divides the display into a grid of cells. Most objects are positioned based on cell coordinates. If the display is a text mode display, *cellWidth* is 1. Otherwise, *cellWidth* is a pixel value set in the display class constructor based on the width of characters in the font.
- *cellHeight* is the height of a cell. OpenZinc logically divides the display into a grid of cells. Most objects are positioned based on cell coordinates. If the display is a text mode display, *cellHeight* is 1. Otherwise, *cellHeight* is a pixel value set in the display class constructor based on the height of characters in the font.
- *preSpace* denotes the size (in pixels) of the white space between the top of an object and the top of a cell.
- *postSpace* denotes the size (in pixels) of the white space between the bottom of an object and the bottom of a cell.
- *miniNumeratorX* and *miniDenominatorX* are values used to determine the width of a minicell. A minicell is a fraction of a cell. By using minicells to size or position

objects instead of cells, more precise placement can be achieved. By default, *miniNumeratorX* is 1 and *miniDenominatorX* is 10. These values result in a minicell being 1/10th the width of a normal cell.

*miniNumeratorY* and *miniDenominatorY* are values used to determine the height of a minicell. A minicell is a fraction of a cell. By using minicells to size or position objects instead of cells, more precise placement can be achieved. By default, *miniNumeratorY* is 1 and *miniDenominatorY* is 10. These values result in a minicell being 1/10th the height of a normal cell.

*operatingSystem* identifies the operating system the application is running on. For example, when running in DOS, *operatingSystem* is "DOS." In MS-Windows it is "Windows."

*windowingSystem* identifies the windowing system the application is using. For example, when using the UI\_GRAPHICS\_DISPLAY class, *windowingSystem* is "GFX." In MS-Windows, it is "Windows."

*backgroundPalette* is a pointer to the color palette used to draw the background of the screen.

*xorPalette* is a pointer to the color palette used when doing XOR drawing.

*colorMap* is a palette table containing basic colors and their appropriate black-and-white or grayscale equivalents.

*markPalette* is a palette table used to draw marked portions of editable fields. This member is only available in DOS and Curses.

*hInstance* is a handle that identifies the current instance of the application. This member is only available in Windows.

*hPrevInstance* indicates if the current instance of the application is the first instance. This member is only available in Windows.

*nCmdShow* is a pointer to any parameters entered from the command line. This member is only available in Windows.

*hab* is the OS/2 anchor block handle. This member is only available in OS/2.

*appContext* is the Xt Intrinsics application context. This member is only available in Motif.

- *topShell* is the initial application shell instance returned by **XtAppInitialize()**. This member is only available in Motif.
- *xDisplay* is the X Window display. This member is only available in Motif.
- *xScreen* is a pointer to the X Window screen. This member is only available in Motif.
- *xScreenNumber* is the X Window screen number. This member is only available in Motif.
- *xGc* is the X Window graphics context. This member is only available in Motif.
- *xorGC* is the X Window graphics context used when doing XOR drawing. This member is only available in Motif.
- *appClass* is the application class name used in the call to *Xtlnitialize*. This member is set to "ZincApp" by default but can be set to whatever the programmer wishes. X Windows looks for a resource file by the same name. This resource file can be used to set default values for fonts, colors and other values. OpenZinc distributes a file called **ZincApp** that contains the default values for a OpenZinc application. The programmer may alter this file as desired. This member is only available in Motif.
- *interleaveStipple* is a Pixmap specifying the interleave fill pattern. This member is only available in Motif.
- *UI\_DISPLAY\_IMAGE* is a structure used when drawing device images. It maintains the following: a region where the device image is located; the device image itself; an image of the screen before the device image is placed there; and a scratch pad used when getting or putting images from and to the screen.

*region* is the region of the images being maintained.

*image* is the device image.

*screen* is a "picture" of the screen at the region specified by *region* before the device image was displayed in the region. This image is used to restore the display after the device image is moved to another location.

*backup* is a "scratch pad" used when transferring images to and from the screen.

- *displayImage* is an array of UI\_DISPLAY\_IMAGE structures. This array maintains the information used to restore the region of the screen where a device image, such as the mouse pointer, has been displayed.
- *codeSet* identifies the code set the application is using. This is used to map characters between the hardware code set and the code set used internally by OpenZinc (e.g., either ISO8859-1 or Unicode).

## UI\_DISPLAY::UI\_DISPLAY

---

### Syntax

```
#include <ui_dsp.hpp>
```

```
UI_DISPLAY(int isText,
           const ZIL_ICHAR *operatingSystem = ZIL_NULLP(ZIL_ICHAR),
           const ZIL_ICHAR *windowingSystem = 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 advanced constructor creates a new UI\_DISPLAY object. The UI\_DISPLAY constructor is protected, since only derived instances of UI\_DISPLAY should be created.

- *isText<sub>in</sub>* indicates whether a text or graphics display is being created. *isText* is TRUE if a text mode display is being created. Otherwise, it is FALSE. *isText* is used to set the *isText* member variable.
- *operatingSystem<sub>in</sub>* identifies the operating system the application is running on. For example, when running in DOS, *operatingSystem* is "DOS." In MS-Windows it is "Windows."

- *windowingSystem<sub>in</sub>* identifies the windowing system the application is using. For example, when using the `UI_GRAPHICS_DISPLAY` class, *windowingSystem* is "GFX." In MS-Windows, it is "Windows."

### Example 1

```
#include <ui_win.hpp>

main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_MSC_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;
    UI_WINDOW_MANAGER *windowManager =
        new UI_WINDOW_MANAGER(display, eventManager);

    // Restore the system,
    delete windowManager;
    delete eventManager;
    delete display;
    return (0);
}
```

### Example 2

```
#include <ui_dsp.hpp>

UI_MSC_DISPLAY::UI_MSC_DISPLAY(int mode):
    UI_DISPLAY(FALSE)
{

}

}
```

## UI\_DISPLAY::~~UI\_DISPLAY

### Syntax

```
#include <ui_dsp.hpp>

virtual ~UI_DISPLAY(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 `UI_DISPLAY` object.

## Example

```
#include <ui_win.hpp>

main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_TEXT_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;
    UI_WINDOW_MANAGER *windowManager = UI_WINDOW_MANAGER(display, eventManager);

    // Restore the system,
    delete windowManager;
    delete eventManager;
    delete display;
    return (0);
}
```

## UI\_DISPLAY::Bitmap

### Syntax

```
#include <ui_dsp.hpp>
```

```
virtual void Bitmap(ZIL_SCREENID screenID, int column, int line, int bitmapWidth,
    int bitmapHeight, const ZIL_UINT8 *bitmapArray,
    const UI_PALETTE *palette = ZIL_NULLP(UI_PALETTE),
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
    ZIL_BITMAP_HANDLE *colorBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE),
    ZIL_BITMAP_HANDLE *monoBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE));
```

## 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 draws a bitmap image. The bitmap is defined by either an array of **ZIL\_UINT8** values, where each array element represents a bitmap pixel color, or by environment-specific bitmap handles.

- *screenID<sub>in</sub>* is the screenID of the object in whose region the drawing should take place. Each object has a screenID that identifies it. The screenID is used to ensure that drawing takes place only on those parts of the screen where the object identified by *screenID* is visible. In addition to objects' screenID's, there are two special screenID values that can be used:

**ID\_DIRECT**—Causes drawing to be done directly to the display, no matter where the drawing location is. This screenID may not have any effect in graphical operating systems since drawing outside the application's window regions may be restricted by the operating system.

**ID\_SCREEN**—Causes drawing to be done to the screen background. Clipping will be performed so that the drawing does not overlap any windows. This screenID may not have any effect in graphical operating systems since drawing outside the application's window regions may be restricted by the operating system.

- *column<sub>in</sub>* and *line<sub>in</sub>* is the upper-left corner of the bitmap, in pixel coordinates, relative to the upper-left corner of the region identified by the *screenID* that was passed in.
- *bitmapWidth<sub>in</sub>* and *bitmapHeight<sub>in</sub>* are the bitmap's pixel width and height.
- *bitmapArray<sub>in</sub>* is the bitmap pattern to be displayed. The bitmap pattern is mapped into an internal palette map (shown below). The color mapping is done to ensure that the bitmap can be shown as clearly as possible if the display is a monochrome or black-and-white display.
- *palette<sub>in</sub>* is a pointer to a palette table that overrides the default palette map. The default palette map is shown below:

```

static UI_PALETTE_colorMap[16] =
{
    { ' ', attrib(BLACK, BLACK), attrib(MONO_BLACK, MONO_BLACK),
      PTN_SOLID_FILL, BLACK, BLACK, BW_BLACK, BW_BLACK, GS_BLACK,
      GS_BLACK },
    { ' ', attrib(BLUE, BLUE), attrib(MONO_DIM, MONO_DIM),
      PTN_SOLID_FILL, BLUE, BLUE, BW_BLACK, BW_BLACK, GS_GRAY,
      GS_GRAY },
    { ' ', attrib(GREEN, GREEN), attrib(MONO_DIM, MONO_DIM),
      PTN_SOLID_FILL, GREEN, GREEN, BW_BLACK, BW_BLACK, GS_GRAY,
      GS_GRAY },
    { ' ', attrib(CYAN, CYAN), attrib(MONO_DIM, MONO_DIM),
      PTN_SOLID_FILL, CYAN, CYAN, BW_BLACK, BW_BLACK, GS_GRAY,
      GS_GRAY },
    { ' ', attrib(RED, RED), attrib(MONO_DIM, MONO_BLACK),
      PTN_SOLID_FILL, RED, RED, BW_BLACK, BW_BLACK, GS_GRAY,
      GS_GRAY },
    { ' ', attrib(MAGENTA, MAGENTA), attrib(MONO_DIM, MONO_DIM),
      PTN_SOLID_FILL, MAGENTA, MAGENTA, BW_BLACK, BW_BLACK, GS_GRAY,
      GS_GRAY },
    { ' ', attrib(BROWN, BROWN), attrib(MONO_DIM, MONO_DIM),
      PTN_SOLID_FILL, BROWN, BROWN, BW_BLACK, BW_BLACK, GS_GRAY,
      GS_GRAY },
    { ' ', attrib(LIGHTGRAY, LIGHTGRAY), attrib(MONO_DIM, MONO_DIM),
      PTN_SOLID_FILL, LIGHTGRAY, LIGHTGRAY, BW_BLACK, BW_BLACK,
      GS_GRAY, GS_GRAY },
    { ' ', attrib(DARKGRAY, DARKGRAY), attrib(MONO_DIM, MONO_DIM),
      PTN_SOLID_FILL, DARKGRAY, DARKGRAY, BW_BLACK, BW_BLACK, GS_GRAY,
      GS_GRAY },
    { ' ', attrib(LIGHTBLUE, LIGHTBLUE), attrib(MONO_NORMAL, MONO_DIM),
      PTN_SOLID_FILL, LIGHTBLUE, LIGHTBLUE, BW_WHITE, BW_WHITE,
      GS_WHITE, GS_WHITE },
    { ' ', attrib(LIGHTGREEN, LIGHTGREEN), attrib(MONO_NORMAL,
      MONO_DIM), PTN_SOLID_FILL, LIGHTGREEN, LIGHTGREEN, BW_WHITE,
      BW_WHITE, GS_WHITE, GS_WHITE },
    { ' ', attrib(LIGHTCYAN, LIGHTCYAN), attrib(MONO_NORMAL, MONO_DIM),
      PTN_SOLID_FILL, LIGHTCYAN, LIGHTCYAN, BW_WHITE, BW_WHITE,
      GS_WHITE, GS_WHITE },
    { ' ', attrib(LIGHTRED, LIGHTRED), attrib(MONO_NORMAL, MONO_DIM),
      PTN_SOLID_FILL, LIGHTRED, LIGHTRED, BW_WHITE, BW_WHITE,
      GS_WHITE, GS_WHITE },
    { ' ', attrib(LIGHTMAGENTA, LIGHTMAGENTA), attrib(MONO_NORMAL,
      MONO_DIM), PTN_SOLID_FILL, LIGHTMAGENTA, LIGHTMAGENTA,
      BW_WHITE, BW_WHITE, GS_WHITE, GS_WHITE },
    { ' ', attrib(YELLOW, YELLOW), attrib(MONO_NORMAL, MONO_DIM),
      PTN_SOLID_FILL, YELLOW, YELLOW, BW_WHITE, BW_WHITE, GS_WHITE,
      GS_WHITE },
    { ' ', attrib(WHITE, WHITE), attrib(MONO_NORMAL, MONO_DIM),
      PTN_SOLID_FILL, WHITE, WHITE, BW_WHITE, BW_WHITE, GS_WHITE,
      GS_WHITE }
};

```

NOTE: If a palette map is provided it must contain entries for all possible bitmap colors.

*clipRegion<sub>in</sub>* is a region that specifies an additional clipping boundary (in addition to the boundary automatically determined by *screenID*) for the **Bitmap( )** function. If *clipRegion* is NULL, no additional clipping is performed.

*colorBitmap<sub>in</sub>* is a ZIL\_BITMAP\_HANDLE structure that is specific to the native environment. *colorBitmap* is the bitmap image to be displayed.

- *monoBitmap<sub>in</sub>* is a ZIL\_BITMAP\_HANDLE 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. Those parts of the bitmap that are not drawn will appear transparent.

**NOTE:** Bitmaps do not have text screen equivalents. Thus, this function should be used with caution.

## Example

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

EVENT_TYPE UIW_ICON::Event(const UI_EVENT &event)
{
    // Switch on the event type,
    int redisplay = FALSE;
    int border = FlagSet(woFlags, WOF_BORDER) ? 1 : 0;
    EVENT_TYPE ccode = UI_WINDOW_OBJECT::LogicalEvent(event, ID_ICON);
    switch (ccode)
    {

    }

    // Redisplay the object information.
    if (redisplay && ratioWidth == 1 && !display->istext)
        display->Bitmap(screenID, iconRegion.left, iconRegion.top, bitmapWidth,
            bitmapHeight, bitmapArray);

    // Return the control code,
    return (ccode);
}
```

## UI\_DISPLAY::BitmapArrayToHandle

### Syntax

```
#include <ui_dsp.hpp>

virtual void BitmapArrayToHandle(ZIL_SCREENID screenID, int bitmapWidth,
    int bitmapHeight, const ZIL_UINT8 *bitmapArray,
    const UI_PALETTE *palette, ZIL_BITMAP_HANDLE *colorBitmap,
    ZIL_BITMAP_HANDLE *monoBitmap);
```

## 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 converts a bitmap array to two handles. Handles are pointers to environment specific storage structures that allow the bitmap to be drawn much faster than drawing individual pixels. The bitmap is defined by an array of **ZIL\_UINT8** values where each array element represents a bitmap pixel color.

- *screenID<sub>in</sub>* is the screenID of the object in whose region the drawing should take place. Each object has a screenID that identifies it. The screenID is used to ensure that drawing takes place only on those parts of the screen where the object identified by *screenID* is visible. In addition to objects' screenID's, there are two special screenID values, **ID\_DIRECT** and **ID\_SCREEN**, that can be used. See the description of the **Bitmap()** function for details.
- *bitmapWidth<sub>in</sub>* and *bitmapHeight<sub>in</sub>* are the bitmap's pixel width and height.
- *bitmapArray<sub>in</sub>* is the bitmap pattern to be converted. The bitmap pattern is mapped into an internal palette map. The color mapping is done to ensure that the bitmap can be shown as clearly as possible if the display is a monochrome or black-and-white display.
- *palette<sub>in</sub>* is a pointer to a palette table that overrides the default palette map.
- *colorBitmap<sub>out</sub>* is a **ZIL\_BITMAP\_HANDLE** structure that is specific to the native environment. *colorBitmap* is the bitmap image to be displayed.
- *monoBitmap<sub>out</sub>* is a **ZIL\_BITMAP\_HANDLE** 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. Those parts of the bitmap that are not drawn will appear transparent.

**NOTE:** Bitmaps do not have text screen equivalents. Thus, this function should be used with caution.

## Example

```
#include <ui_dsp.hpp>

void UI_OS2_DISPLAY::Bitmap(ZIL_SCREENID screenID, int left, int top,
    int bitmapWidth, int bitmapHeight, const ZIL_UINT8 *bitmapArray,
    const UI_PALETTE *palette, const UI_REGION *,
    ZIL_BITMAP_HANDLE *_colorBitmap, ZIL_BITMAP_HANDLE *_monoBitmap)
{

    // Convert the bitmap array then draw the bitmap,
    if (!colorBitmap)
        BitmapArrayToHandle(screenID, bitmapWidth, bitmapHeight, bitmapArray,
            palette, &colorBitmap, &monoBitmap);
```

## UI\_DISPLAY::BitmapHandleTo Array\_\_\_\_\_

### Syntax

```
#include <ui_dsp.hpp>

virtual void BitmapHandleToArray(ZIL_SCREENID screenID,
    ZIL_BITMAP_HANDLE colorBitmap, ZIL_BITMAP_HANDLE monoBitmap,
    int *bitmapWidth, int *bitmapHeight, ZIL_UINT8 **bitmapArray);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function converts an environment-specific bitmap handle to an array of **ZIL\_UINT8** values where each array element represents a bitmap pixel color.

- *screenID<sub>in</sub>* is the screenID of the object in whose region the drawing should take place. Each object has a screenID that identifies it. The screenID is used to ensure that drawing takes place only on those parts of the screen where the object identified

by *screenID* is visible. In addition to objects' *screenID*'s there are two special *screenID* values, *ID\_DIRECT* and *ID\_SCREEN*, that can be used. See the description of the **Bitmap()** function for details.

- *colorBitmap<sub>in</sub>* is a *ZIL\_BITMAP\_HANDLE* structure that is specific to the native environment. *colorBitmap* is the bitmap image to be displayed.
- *monoBitmap<sub>in</sub>* is a *ZIL\_BITMAP\_HANDLE* 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. Those parts of the bitmap that are not drawn will appear transparent.
- *bitmapWidth<sub>out</sub>* and *bitmapHeight<sub>out</sub>* are the bitmap's pixel width and height.
- *bitmapArray<sub>out</sub>* is the bitmap pattern that was converted.

## UI\_DISPLAY::Ellipse

### Syntax

```
#include <ui_dsp.hpp>
```

```
virtual void Ellipse(ZIL_SCREENID screenID, int column, int line, int startAngle,  
int endAngle, int xRadius, int yRadius, const UI_PALETTE *palette,  
int fill = FALSE, int _xor = FALSE,
```

```
const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
```

### 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 draws and or fills an ellipse. The ellipse is defined by starting and ending angles and horizontal and vertical radii.

- *screenID<sub>in</sub>* is the screenID of the object in whose region the drawing should take place. Each object has a screenID that identifies it. The screenID is used to ensure that drawing takes place only on those parts of the screen where the object identified by *screenID* is visible. In addition to objects' screenID's there are two special screenID values, ID\_DIRECT and ID\_SCREEN, that can be used. See the description of the **Bitmap()** function for details.
- *column<sub>in</sub>* and *line<sub>in</sub>* is the upper-left corner of the bitmap, in pixel coordinates, relative to the upper-left corner of the region identified by the *screenID* that was passed in.
- *startAngle<sub>in</sub>* and *endAngle<sub>in</sub>* are starting and ending angles, in degrees, of the ellipse. If a complete ellipse is desired, *startAngle* should be 0 and *endAngle* should be 360.
- *xRadius<sub>in</sub>* and *yRadius<sub>in</sub>* are the horizontal and vertical radii of the ellipse, in pixels.
- *palette<sub>in</sub>* is a pointer to the palette structure that defines the color to draw the ellipse. The palette's foreground color is used to draw the border of the ellipse. The palette's background color is used to fill the ellipse (if *fill* is TRUE).
- *fill<sub>in</sub>* indicates whether the ellipse should be filled. If *fill* is TRUE, the ellipse is filled according to the specified palette's fill pattern and background color. Otherwise the ellipse is not filled.
- *-Xor<sub>in</sub>* indicates if the ellipse should be XOR'ed with the image it overwrites. If *\_xor* is TRUE, the ellipse is drawn using an XOR attribute. Otherwise it simply draws over the existing image.
- *clipRegion<sub>in</sub>* is a region that specifies an additional clipping boundary (in addition to the boundary automatically determined by *screenID*) for the **Ellipse( )** function. If *clipRegion* is NULL, no additional clipping is performed.

**NOTE:** Ellipses do not have text screen equivalents. Thus, this function should be used with caution.

## Example

```
#include <ui_dsp.hpp>

class CIRCLE : public UI_WINDOW_OBJECT
{
public:
```



```

    int fill;
} ;

EVENT_TYPE CIRCLE::Event(const UI_EVENT &event)
{
    EVENT_TYPE ccode = UI_WINDOW_OBJECT::LogicalEvent(event, ID_CIRCLE);

    switch (ccode)
    {
    case S_DISPLAY_INACTIVE:
    case S_DISPLAY_ACTIVE:
        if (display->isText || !UI_WINDOW_OBJECT::NeedsUpdate(event, ccode))
            break;

        // Set up a temporary clip region then draw the circle.
        display->RegionDefine(screenID I 0x1000, trueRegion);
        int column = trueRegion.left + (relative.right - relative.left) / 2;
        int line = trueRegion.top + (relative.bottom - relative.top) / 2;
        int radius = (relative.bottom - relative.top) / 2;
        if (radius < (relative.right - relative.left) / 2)
            radius = (relative.right - relative.left) / 2;
        display->Ellipse(screenID, column, line, 0, 360, radius,
            radius, lastPalette, fill);
        // Restore the normal region.
        display->RegionDefine(screenID, trueRegion);
        break;

    }

    // Return the control code,
    return (ccode);
}

```

## UI\_DISPLAY::IconArrayToHandle

### Syntax

```

#include <ui_dsp.hpp>

virtual void IconArrayToHandle(ZIL_SCREENID screenID, int iconWidth, int iconHeight,
    const ZIL_UINT8 *iconArray, const UI_PALETTE *palette,
    ZIL_ICON_HANDLE *icon)\

```

### 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 converts an icon image array to an icon handle. A handle is a pointer to an environment specific storage structure that allows the icon to be drawn much faster than drawing individual pixels. The bitmap is defined by an array of **ZIL\_UINT8** values where each array element represents a pixel color.

- *screenID<sub>in</sub>* is the screenID of the object in whose region the drawing should take place. Each object has a screenID that identifies it. The screenID is used to ensure that drawing takes place only on those parts of the screen where the object identified by *screenID* is visible. In addition to objects' screenID's there are two special screenID values, ID\_DIRECT and ID\_SCREEN, that can be used. See the description of the **Bitmap()** function for details.
- *iconWidth<sub>in</sub>* and *iconHeight<sub>in</sub>* are the icon's pixel width and height.
- *iconArray<sub>in</sub>* is the icon image pattern. The icon image pattern is mapped into an internal palette map. The color mapping is done to ensure that the image can be shown as clearly as possible if the display is a monochrome or black-and-white display.
- *palette<sub>in</sub>* is a pointer to a palette table that overrides the default palette map.
- *icon<sub>out</sub>* 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.

**NOTE:** Icons do not have text screen equivalents. Thus, this function should be used with caution.

## UI\_DISPLAY::IconHandleToArray

### Syntax

```
#include <ui_dsp.hpp>
```

```
virtual void IconHandleToArray(ZIL_SCREENID screenID, ZIL_ICON_HANDLE icon,  
int *iconWidth, int *iconHeight, ZIL_UINT8 **iconArray);
```

## 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 converts an icon handle to an array of **ZIL\_UINT8** values where each array element represents an icon image pixel color.

- *screenID<sub>in</sub>* is the screenID of the object in whose region the drawing should take place. Each object has a screenID that identifies it. The screenID is used to ensure that drawing takes place only on those parts of the screen where the object identified by *screenID* is visible. In addition to objects' screenID's there are two special screenID values, ID\_DIRECT and ID\_SCREEN, that can be used. See the description of the **Bitmap()** function for details.
- *icon<sub>in</sub>* is an operating system-specific icon handle that is to be converted. 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.
- *iconWidth<sub>out</sub>* and *iconHeight<sub>out</sub>* are the pixel width and height of the icon array.
- *iconArray<sub>out</sub>* is the icon image pattern that was converted.

NOTE: Icons do not have text screen equivalents. Thus, this function should be used with caution.

## UI\_DISPLAY::Line

### Syntax

```
#include <ui_dsp.hpp>
```

```
virtual void Line(ZIL_SCREENID screenID, int column1, int line1, int column2, int line2,  
    const UI_PALETTE *palette, int width = 1, int _xor = FALSE,  
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
```

## 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 draws a line. The line is defined by a starting point and an ending point. Care should be taken when using this function in text mode, as diagonal lines will not display as expected.

- *screenID<sub>in</sub>* is the screenID of the object in whose region the drawing should take place. Each object has a screenID that identifies it: The screenID is used to ensure that drawing takes place only on those parts of the screen where the object identified by *screenID* is visible. In addition to objects' screenID's there are two special screenID values, ID\_DIRECT and ID\_SCREEN, that can be used. See the description of the **Bitmap()** function for details.
- *column1<sub>in</sub>* and *line1<sub>in</sub>* is the starting position of the line relative to the upper-left corner of the region identified by the *screenID* that was passed in. These values should be in pixel coordinates if the display is a graphics display or in text coordinates if it is a text mode display.
- *column2<sub>in</sub>* and *line2<sub>in</sub>* is the ending position of the line relative to the upper-left corner of the region identified by the *screenID* that was passed in. These values should be in pixel coordinates if the display is a graphics display or in text coordinates if it is a text mode display.
- *palette<sub>in</sub>* is a pointer to the palette structure that defines the color to draw the line. The palette's foreground color is used to draw the line.
- *width<sub>in</sub>* is the width of the line. If the application is running in text mode, *width* is in cell widths. Otherwise, *width* is in pixel coordinates.
- *\_xor<sub>in</sub>* indicates if the line should be XOR'ed with the image it overwrites. If *\_xor* is TRUE, the line is drawn using an XOR attribute. Otherwise it simply draws over the existing image.

- *clipRegion<sub>m</sub>* is a region that specifies an additional clipping boundary (in addition to the boundary automatically determined by *screenID*) for the **Line( )** function. If *clipRegion* is NULL, no additional clipping is performed.

## Example

```
void UI_WINDOW_OBJECT::Border(EVENT_TYPE ccode, UI_REGION &region,
    const UI_PALETTE *palette)
{
    // Determine the border and update the region,
    region=trueRegion;

    int displayBorder = (ccode == S_DISPLAY_ACTIVE ||
        ccode == S_NON_CURRENT || ccode == S_DISPLAY_INACTIVE ||
        ccode == S_CURRENT) ? TRUE : FALSE;

    if (displayBorder && palette)
    {
        UI_PALETTE tPalette = *palette;
        tPalette.colorForeground = tPalette.colorBackground;
        tPalette.bwForeground = tPalette.bwBackground;
        tPalette.grayScaleForeground = tPalette.grayScaleBackground;
        display->Line(screenID, region.left, region.top, region.left,
            region.bottom, tPalette);
        display->Line(screenID, region.right, region.top, region.right,
            region.bottom, tPalette);
    }
    region.left++;
    region.right--;
```

## UI\_DISPLAY::MapColor

### Syntax

```
#include <ui_dsp.hpp>

virtual ZIL_COLOR MapColor(const UI_PALETTE *palette, int isForeground);
```

## 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 returns the mapped color from a palette according to the type of display and the programmer request. For example, if the display is a black-and-white display, the appropriate black-and-white color will be returned.

- *returnValue<sub>out</sub>* is the color that was mapped to, according to the type of display and the type of request.
- *palette<sub>in</sub>* is the palette from which the colors are to be mapped.
- *isForeground<sub>in</sub>* indicates whether the palette's foreground or background color is desired. If *isForeground* is TRUE, the foreground color will be returned. Otherwise, the background color will be returned.

## UI\_DISPLAY::Polygon

### Syntax

```
#include <ui_dsp.hpp>
```

```
virtual void Polygon(ZIL_SCREENID screenID, int numPoints, const int *polygonPoints,  
    const UI_PALETTE *palette, int fill = FALSE, int _xor = FALSE,  
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
```

### 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 draws and/or fills a polygon. The polygon is defined by a set of vertices.

- *screenID<sub>in</sub>* is the screenID of the object in whose region the drawing should take place. Each object has a screenID that identifies it. The screenID is used to ensure that drawing takes place only on those parts of the screen where the object identified by *screenID* is visible. In addition to objects' screenID's there are two special screenID values, ID\_DIRECT and ID\_SCREEN, that can be used. See the description of the **Bitmap()** function for details.
- *numPoints<sub>in</sub>* is the number of points in the polygon.
- *polygonPoints<sub>in</sub>* is a pointer to an array of integers (i.e., *numPoints* x 2). Each integer pair gives a column and line point on the polygon. These values should be in pixel coordinates relative to the upper-left corner of the region identified by the *screenID* that was passed in.
- *palette<sub>in</sub>* is a pointer to the palette structure that defines the color to draw the polygon. The palette's foreground color is used to draw the border of the polygon. The palette's background color is used to fill the polygon (if *fill* is TRUE).
- *fill<sub>in</sub>* indicates whether the polygon should be filled. If *fill* is TRUE, the polygon is filled according to the specified palette's fill pattern and background color. Otherwise the polygon is not filled.
- *\_xor<sub>in</sub>* indicates if the polygon should be XOR'ed with the image it overwrites. If *\_xor* is TRUE, the polygon is drawn using an XOR attribute. Otherwise it simply draws over the existing image.
- *clipRegion<sub>in</sub>* is a region that specifies an additional clipping boundary (in addition to the boundary automatically determined by *screenID*) for the **Polygon()** function. If *clipRegion* is NULL, no additional clipping is performed.

NOTE: Polygons do not have text screen equivalents. Thus, this function should be used with caution.

## Example

```
#include <ui_dsp.hpp>

class TRIANGLE : public UI_WINDOW_OBJECT
```

```

public:

    int fill;
};

EVENT_TYPE TRIANGLE::Event(const UI_EVENT &event)
{
    EVENT_TYPE ccode = UI_WINDOW_OBJECT::LogicalEvent(event, ID_TRIANGLE);
    switch (ccode)
    {
        case S_DISPLAY_INACTIVE:
        case S_DISPLAY_ACTIVE:
            if (display->isText || !UI_WINDOW_OBJECT::NeedsUpdate(event, ccode))
                break;

            // Set up a temporary clip region then draw the triangle.
            display->RegionDefine (screenID 0x1000, trueRegion);
            int triangle[8];
            triangle[0] = triangle[6] =
            trueRegion.left + (relative.right - relative.left) / 2;
            triangle[1] = triangle[7] = trueRegion.top;
            triangle[2] = trueRegion.left;
            triangle[3] = triangle[5] = trueRegion.top + relative.bottom;
            triangle[4] = trueRegion.left + relative.right;
            display->Polygon(screenID, 4, triangle, lastPalette, fill);
            // Restore the normal region.
            display->RegionDefine (screenID, trueRegion);
            break;

    }

    // Return the control code,
    return (ccode);
}

```

## UI\_DISPLAY: .-Rectangle

### Syntax

```
#include <ui_dsp.hpp>
```

```
void Rectangle(ZIL_SCREENID screenID, const UI_REGION &region,
    const UI_PALETTE *palette, int width = 1, int fill = FALSE, int _xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
or
```

```
virtual void Rectangle(ZIL_SCREENID screenID, int left, int top, int right, int bottom,
    const UI_PALETTE *palette, int width = 1, int fill = FALSE, int _xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
```



## 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 draw and/or fill a bar or rectangle.

The first virtual function draws a rectangular box determined by the region specified.

- *screenID<sub>in</sub>* is the screenID of the object in whose region the drawing should take place. Each object has a screenID that identifies it. The screenID is used to ensure that drawing takes place only on those parts of the screen where the object identified by *screenID* is visible. In addition to objects' screenID's there are two special screenID values, ID\_DIRECT and ID\_SCREEN, that can be used. See the description of the **Bitmap()** function for details.
- *region<sub>in</sub>* is the region that defines the coordinates of the rectangle. The region should be specified in pixels if in graphics mode or in screen coordinates if in text mode. The region is relative to the upper-left corner of the region identified by the *screenID* that was passed in.
- *palette<sub>in</sub>* is a pointer to the palette structure that defines the color to draw the rectangle. The palette's foreground color is used to draw the border of the rectangle. The palette's background color is used to fill the rectangle (if *fill* is TRUE).
- *width* specifies the width of the rectangle's border. If the application is running in text mode, *width* is in cell widths. Otherwise, *width* is in pixel coordinates.
- *fill<sub>in</sub>* indicates whether the rectangle should be filled. If *fill* is TRUE, the rectangle is filled according to the specified palette's fill pattern and background color. Otherwise the rectangle is not filled.
- *\_xor<sub>in</sub>* indicates if the rectangle should be XOR'ed with the image it overwrites. If *\_xor* is TRUE, the rectangle is drawn using an XOR attribute. Otherwise it simply draws over the existing image.

- *clipRegion<sub>in</sub>* is a region that specifies an additional clipping boundary (in addition to the boundary automatically determined by *screenID*) for the **Rectangle()** function. If *clipRegion* is NULL, no additional clipping is performed.

The second virtual function draws a rectangular box defined by the corners specified.

- *screenID<sub>in</sub>* is the screenID of the object in whose region the drawing should take place. Each object has a screenID that identifies it. The screenID is used to ensure that drawing takes place only on those parts of the screen where the object identified by *screenID* is visible. In addition to objects' screenID's there are two special screenID values, ID\_DIRECT and ID\_SCREEN, that can be used. See the description of the **Bitmap()** function for details.
- *lef<sub>in</sub>* and *top<sub>in</sub>* is the starting position of the rectangle relative to the upper-left corner of the region identified by the *screenID* that was passed in. These values should be in pixel coordinates if the display is a graphics display or in text coordinates if it is a text mode display.
- *right<sub>in</sub>* and *bottom<sub>in</sub>* are the ending position of the rectangle relative to the upper-left corner of the region identified by the *screenID* that was passed in. These values should be in pixel coordinates if the display is a graphics display or in text coordinates if it is a text mode display.
- *palette<sub>in</sub>* is a pointer to the palette structure that defines the color to draw the rectangle. The palette's foreground color is used to draw the border of the rectangle. The palette's background color is used to fill the rectangle (if *fill* is TRUE).
- *width<sub>in</sub>* specifies the width of the rectangle's border. If the application is running in text mode, *width* is in cell widths. Otherwise, *width* is in pixel coordinates.
- *fill<sub>in</sub>* indicates whether the rectangle should be filled. If *fill* is TRUE, the rectangle is filled according to the specified palette's fill pattern and background color. Otherwise the rectangle is not filled.
- *\_jcor<sub>in</sub>* indicates if the rectangle should be XOR'ed with the image it overwrites. If *\_xor* is TRUE, the rectangle is drawn using an XOR attribute. Otherwise it simply draws over the existing image.
- *clipRegion<sub>in</sub>* is a region that specifies an additional clipping boundary (in addition to the boundary automatically determined by *screenID*) for the **Rectangle()** function. If *clipRegion* is NULL, no additional clipping is performed.

## Example

```
#include <ui_win.hpp>

EVENT_TYPE UIW_BORDER::Event(const UI_EVENT &event)
{
    // Switch on the event type.
    UI_REGION region;
    EVENT_TYPE ccode = UI_WINDOW_OBJECT::LogicalEvent(event, ID_BORDER);
    switch (ccode)
    {
    case S_DISPLAY_INACTIVE:
    case S_DISPLAY_ACTIVE:
        // Draw the borders around the object.
        if (!display->isText && !UI_WINDOW_OBJECT::NeedsUpdate(event, ccode))
            break;
        UI_PALETTE *palette = UI_WINDOW_OBJECT::LogicalPalette(ccode);
        lastPalette = palette;
        if (display->isText)
            display->Rectangle(screenID, trueRegion, palette,
                               (ccode == S_DISPLAY_ACTIVE) ? 2 : 1);
        else
        {
            region = parent->trueRegion;
            eventManager->DevicesHide(parent->trueRegion);
            UI_PALETTE *outlinePalette = MapPalette(paletteMapTable,
                                                    PM_ACTIVE, ID_BLACK_SHADOW);
            display->Rectangle(screenID, trueRegion, outlinePalette);
            display->Rectangle(screenID, region, outlinePalette);
            UI_WINDOW_OBJECT::Shadow(region, 1);

            // Display the top and bottom lines,
            int temp = region.bottom;
            region.bottom = trueRegion.top - 1;
            display->Rectangle(screenID, region, palette, 0, TRUE);
            region.bottom = temp;

            temp = region.top;
            region.top = trueRegion.bottom + 1;
            display->Rectangle(screenID, region, palette, 0, TRUE);
            region.top = temp;
        }
    }
    // Return the control code,
    return (ccode);
}
```

## UI\_DISPLAY::RectangleXORPiff

---

### Syntax

```
#include <ui_dsp.hpp>

virtual void RectangleXORDiff(const UI_REGION &oldRegion,
                             const UI_REGION &newRegion, ZIL_SCREENID screenID = ID_SCREEN,
                             const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
```

## Portability

This function is available on the following environments:

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

## Remarks

This advanced virtual function draws an XOR rectangle at two locations: an original region and a new region. An example of when this function may be necessary is when dragging the thumb button of a scroll bar. Often, the original thumb button position is highlighted by drawing an XOR rectangle at that location, and the current location it is being dragged to is also highlighted by an XOR rectangle.

- *oldRegion<sub>in</sub>* defines the original rectangle that is to be drawn.
- *newRegion<sub>in</sub>* defines the new rectangle that is to be drawn.
- *screenID<sub>in</sub>* is the screenID of the object in whose region the drawing should take place. Each object has a screenID that identifies it. The screenID is used to ensure that drawing takes place only on those parts of the screen where the object identified by *screenID* is visible. In addition to objects' screenID's there are two special screenID values, ID\_DIRECT and ID\_SCREEN, that can be used. See the description of the **Bitmap( )** function for details.
- *clipRegion<sub>in</sub>* is a region that specifies an additional clipping boundary (in addition to the boundary automatically determined by *screenID*) for the **RectangleXORDiff()** function. If *clipRegion* is NULL, no additional clipping is performed.

## Example

```
#include <ui_win.hpp>

void UI_WINDOW_MANAGER::Modify(UI_WINDOW_OBJECT *object,
    const UI_EVENT &event)
{
    UI_REGION newRegion = object->trueRegion;
    UI_REGION oldRegion = newRegion;

    // Update the new region,
    if (oldRegion.left != newRegion.left ||
        oldRegion.top != newRegion.top ||
        oldRegion.right != newRegion.right ||
```

```

        oldRegion.bottom != newRegion.bottom)
    {
        // Compute the lower-right coordinates.
        newRegion.right = newRegion.left + width - 1;
        newRegion.bottom = newRegion.top + height - 1;

        // Remove the old region and update the new region,
        if (eventManager->Get(tEvent, Q_NO_BLOCK | Q_NO_DESTROY) != 0 ||
            MapEvent(eventMapTable, tEvent, ID_WINDOW_OBJECT, ID_WINDOW_OBJECT)
                != L_CONTINUE_SELECT)
        {
            display->RectangleXORDiff(oldRegion, newRegion);
            oldRegion = newRegion;
        }
    }
}
}

```

## UI\_DISPLAY::RegionDefine

### Syntax

```
#include <ui_dsp.hpp>
```

```
void RegionDefine(ZIL_SCREENID screenID, const UI_REGION &region);
```

*or*

```
virtual void RegionDefine(ZIL_SCREENID screenID, int left, int top, int right,
    int bottom);
```

### Portability

This function is available on the following environments:

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

### Remarks

These advanced overloaded functions are used to reserve a specified region of the screen for a particular screenID. The region is added to the display's region list. While this function exists for all environments, it performs no action except for DOS and Curses displays.

The first function defines a region determined by the region specified.

- *screenID<sub>in</sub>* is the identification associated with the defined region. Once a region has been defined, only those objects with the same screenID will be allowed to write to that screen region.
- *region<sub>in</sub>* defines the rectangular region to be reserved.

The second virtual function defines a region determined by the corners specified.

- *screenID<sub>in</sub>* is the identification associated with the defined region. Once a region has been defined, only those objects with the same screenID will be allowed to write to that screen region.
- *left<sub>in</sub> top<sub>in</sub> right<sub>in</sub> and bottom<sub>in</sub>* define the rectangular region to be reserved.

## Example

```
#include <ui_win.hpp>

EVENT_TYPE UIW_ICON::Event(const UI_EVENT &event)
{
    // Switch on the event type,
    int redisplay = FALSE;
    int border = FlagSet(woFlags, WOF_BORDER) ? 1 : 0;
    EVENT_TYPE ccode = UI_WINDOW_OBJECT::LogicalEvent(event, ID_ICON);
    switch (ccode)
    {
        case S_DEFINE_REGION:
            if (!parent)
            {
                if (!display->isText)
                    display->RegionDefine(screenID, iconRegion);
                if (string)
                    display->RegionDefine(screenID, stringRegion);
            }
            break;

    }

    // Return the control code,
    return (ccode);
}
```

## **UI\_DISPLAY::RegionInitialize**

### **Syntax**

```
#include <ui_dsp.hpp>
```

```
virtual void RegionInitialize(UI_REGION &region, const UI_REGION *clipRegion,  
    int left, int top, int right, int bottom);
```

### **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 initializes *region* with the region defined by *left*, *top*, *right* and *bottom*.

- *returnValue<sub>out</sub>* indicates if the region has a positive area. *returnValue* is TRUE if the upper-left corner of the region is above and to the left of the lower-right corner. Otherwise, it is FALSE.
- *region<sub>out</sub>* is the region that is to be initialized.
- *clipRegion<sub>in</sub>* is a region that specifies a clipping boundary. The region defined by *left*, *top*, *right* and *bottom* is clipped by *clipRegion* before being placed in *region*. If *clipRegion* is NULL, no additional clipping is performed.
- *left<sub>in</sub>*, *top<sub>in</sub>*, *right<sub>in</sub>* and *bottom<sub>in</sub>* define the rectangular region to be initialized.

## **UI\_DISPLAY::RegionMove**

### **Syntax**

```
#include <ui_dsp.hpp>
```

```
virtual void RegionMove(const UI_REGION &oldRegion, int newColumn, int newLine,
```

```
ZIL_SCREENID oldScreenID = ID_SCREEN,  
ZIL_SCREENID newScreenID = ID_SCREEN);
```

## 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 copies the screen image from one region to another region.

- *oldRegion<sub>in</sub>* is the screen region to be moved.
- *newColumn<sub>in</sub>* and *newLine<sub>in</sub>* is the upper-left corner of the new screen location for the screen image.
- *oldScreenID<sub>in</sub>* is the screenID of the original screen image location.
- *newScreenID<sub>in</sub>* is the screenID of the destination screen location.

## Example

```
#include <ui_win.hpp>  
  
void UI_WINDOW_OBJECT::Modify(UI_WINDOW_OBJECT *object,  
    const UI_EVENT &event)  
{  
    UI_REGION newRegion = object->trueRegion;  
    UI_REGION oldRegion = newRegion;  
  
    // Move the region,  
    if (!sizeObject)  
        display->RegionMove(object->trueRegion, newRegion.left, newRegion.top);
```



## UI\_DISPLAY::Text

### Syntax

```
#include <ui_dsp.hpp>
```

```
virtual void Text(ZIL_SCREENID screenID, int left, int top, const ZIL_ICHAR *text,  
                const UI_PALETTE *palette, int length = -1, int fill = TRUE, int _xor = FALSE,  
                const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),  
                ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
```

### 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 draws a text string.

- *screenID<sub>in</sub>* is the screenID of the object in whose region the drawing should take place. Each object has a screenID that identifies it. The screenID is used to ensure that drawing takes place only on those parts of the screen where the object identified by *screenID* is visible. In addition to objects' screenID's there are two special screenID values, ID\_DIRECT and ID\_SCREEN, that can be used. See the description of the **Bitmap()** function for details.
- *left<sub>in</sub>* and *top<sub>in</sub>* is the starting position of the text relative to the upper-left corner of the region identified by the *screenID* that was passed in. These values should be in pixel coordinates if the display is a graphics display or in text coordinates if it is a text mode display.
- *text<sub>in</sub>* is a pointer to the text that is to be displayed.
- *palette<sub>in</sub>* is a pointer to the palette structure that defines the color to draw the text. The palette's foreground color is used to draw the text. The palette's background color is used to draw the background of the text (if *fill* is TRUE).

- *fill<sub>in</sub>* indicates whether the text background should be filled. If *fill* is TRUE, the text background is filled according to the specified palette's fill pattern and background color. Otherwise the text background is not filled.
- *\_xor<sub>in</sub>* indicates if the text should be XOR'ed with the image it overwrites. If *\_xor* is TRUE, the text is drawn using an XOR attribute. Otherwise it simply draws over the existing image.
- *clipRegion<sub>in</sub>* is a region that specifies an additional clipping boundary (in addition to the boundary automatically determined by *screenID*) for the **Text()** function. If *clipRegion* is NULL, no additional clipping is performed.
- *font<sub>in</sub>* is the font to be used when drawing the text, *font* is an index into the display's *fontTable* array.

## Example

```
#include <ui_win.hpp>

EVENT_TYPE GRAPH::DrawItem(const UI_EVENT &, EVENT_TYPE )
{
    // Virtualize the display;
    display->VirtualGet(screenID, trueRegion);

    display->Text(screenID, trueRegion.left, trueRegion.top+display->cellHeight/2,
        " Item 1", &redPalette, -1, FALSE);
    display->Text(screenID, trueRegion.left, trueRegion.top+display->cellHeight*3/2,
        " Item 2", &bluePalette, -1, FALSE);
    display->Text(screenID, trueRegion.left, trueRegion.top+display->cellHeight*5/2,
        " Item 2", &greenPalette, -1, FALSE);

    // Un-virtualize the display;
    display->VirtualPut(screenID);

    return (TRUE);
}
```

## UI\_DISPLAY::TextHeight

### Syntax

```
#include <ui_dsp.hpp>

virtual int TextHeight(const ZIL_ICHAR *string,
    ZIL_SCREENID screenID = ID_SCREEN,
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
```

## Portability

This virtual function is available on the following environments:

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

## Remarks

This function returns the height of a specified string.

- *returnValue*<sub>out</sub> is the height of the string. If the application is running in text mode, this value is always 1. Otherwise, *returnValue* is the pixel height of the string.
- *string*<sub>in</sub> is a pointer to the string whose height is to be determined.
- *screenID*<sub>in</sub> is the screenID of an object or of the screen (i.e., ID\_SCREEN). In some environments, the screenID is required to calculate the text parameters.
- *font*<sub>in</sub> is the font to be used when measuring the text string.

## Example

```
#include <ui_win.hpp>

void UI_WINDOW_OBJECT::Text(char *string, int depth, int ccode,
    const UI_PALETTE *palette)
{
    // Display the text to the screen.

    // Make sure it is a valid string,
    if (string == 0 || string[0] == '\0')
        return;

    // See if the string will fit.
    int height = display->TextHeight(string);
    if (region.bottom - region.top + 1 < height)
        return;

    char scrapBuffer[128];
    strncpy(scrapBuffer, string, 128);
    scrapBuffer[127] = '\0';
    char *hotKey = strchr(scrapBuffer, '~');
    if (hotKey)
        strcpy(hotKey, hotKey + 1);

    int width = display->TextWidth(scrapBuffer);
    if (width > region.right - region.left + 1)
        {
```

```

        width = region.right - region.left;
        scrapBuffer[width / display->cellwidth] = '\0'
    }

}

```

## UI\_DISPLAY::TextWidth

### Syntax

```

#include <ui_dsp.hpp>

virtual int TextWidth(const ZIL_ICHAR *string,
    ZIL_SCREENID screenID = ID_SCREEN,
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);

```

### 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 width of a specified string.

- *returnValue<sub>out</sub>* is the width of the string. If the application is running in text mode, this value is in screen coordinates. Otherwise, *returnValue* is the pixel width of the string.
- *string<sub>in</sub>* is a pointer to the string whose width is to be determined.
- *screenID<sub>in</sub>* is the screenID of an object or of the screen (i.e., ID\_SCREEN). In some environments, the screenID is required to calculate the text parameters.
- *font<sub>in</sub>* is the font to be used when measuring the text string.

## Example

```
#include <ui_win.hpp>

void UI_WINDOW_OBJECT::Text(char *string, int depth, int ccode,
    const UI_PALETTE *palette)
{
    // Display the text to the screen.

    // Make sure it is a valid string,
    if (string == 0 || string[0] == '\0')
        return;

    // See if the string will fit.
    int height = display->TextHeight(string);
    if (region.bottom - region.top + 1 < height)
        return;

    char scrapBuffer[128];
    strncpy(scrapBuffer, string, 128);
    scrapBuffer[127] = '\0';
    char *hotKey = strchr(scrapBuffer, '~');
    if (hotKey)
        strcpy(hotKey, hotKey + 1);
    int width = display->TextWidth(scrapBuffer);
    if (width > region.right - region.left + 1)
    {
        width = region.right - region.left;
        scrapBuffer[width / display->cellWidth] = '\0';
    }

}
```

## UI\_DISPLAY::VirtualGet

### Syntax

```
#include <ui_dsp.hpp>

int VirtualGet(ZIL_SCREENID screenID, const UI_REGION &region);
    or
virtual int VirtualGet(ZIL_SCREENID screenID, int left, int top, int right, int bottom);
```

## 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 attempt to optimize multiple successive drawing calls. In some environments, all drawing done to a region after a call to **VirtualGet( )** will be performed on a virtual buffer that is not displayed until **VirtualPut()** is called. This function may remove the images of all devices that are within the region specified, thus preventing the device images from having to erase and redraw numerous times. Some environments use this call to obtain information from the operating system that will allow subsequent drawing to be performed properly.

This function must be called before any calls are made to display primitives (e.g., **Rectangle( )**, **Bitmap()**, etc.). The **VirtualPut()** function must be called when done drawing.

The first overloaded function prepares for drawing within the region specified by *region*.

- *returnValue<sub>out</sub>* is always TRUE.
- *screenID<sub>in</sub>* is the screenID of the object in whose region the drawing will be done. Each object has a screenID that identifies it. The screenID is used to ensure that drawing takes place only on those parts of the screen where the object identified by *screenID* is visible. In addition to objects' screenID's there are two special screenID values, ID\_DIRECT and ID\_SCREEN, that can be used. See the description of the **Bitmap()** function for details.
- *region<sub>in</sub>* is the region where drawing is to occur. This region is relative to the upper-left corner of the region identified by the *screenID* that was passed in.

The second overloaded function prepares for drawing within the region specified by *left*, *top*, *right* and *bottom*.

- *returnValue<sub>out</sub>* is always TRUE.
- *screenID<sub>in</sub>* is the screenID of the object in whose region the drawing will be done. Each object has a screenID that identifies it. The screenID is used to ensure that

drawing takes place only on those parts of the screen where the object identified by *screenID* is visible. In addition to objects' *screenID*'s there are two special *screenID* values, `ID_DIRECT` and `ID_SCREEN`, that can be used. See the description of the **Bitmap()** function for details.

- *left<sub>in</sub>*, *top<sub>in</sub>*, *right<sub>in</sub>* and *bottom<sub>in</sub>* is the region where drawing is to occur. This region is relative to the upper-left corner of the region identified by the *screenID* that was passed in.

## Example

```
WO_GRAPH::DrawX(UI_DISPLAY *display)
{
    // Copy the screen into the virtual buffer,
    display->VirtualGet(screenID, 0, 0, 100, 100);

    display->Line(screenID, 0, 0, 100, 100);
    display->Line(screenID, 0, 100, 100, 0);

    // Copy the virtual buffer back to the screen.
    display->VirtualPut(screenID);
}
```

## UI\_DISPLAY::VirtualPut

---

### Syntax

```
#include <ui_dsp.hpp>
```

```
virtual int VirtualPut(ZIL_SCREENID screenID);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function completes the process started with the call to **VirtualGet()**. In those environments that did subsequent drawing to a virtual buffer, this function causes the buffer to be displayed. In those environments whose device images were disabled, they will be enabled.

This function must be called after drawing has been completed.

- *returnValue*<sub>out</sub> is always TRUE.
- *screenID*<sub>in</sub> is the screenID of the object in whose region the drawing was done. Each object has a screenID that identifies it. The screenID is used to ensure that drawing takes place only on those parts of the screen where the object identified by *screenID* is visible. In addition to objects' screenID's there are two special screenID values, ID\_DIRECT and ID\_SCREEN, that can be used. See the description of the **Bitmap( )** function for details.

### Example

```
WO_GRAPH::DrawX(UI_DISPLAY *display)
{
    // Copy the screen into the virtual buffer.
    display->VirtualGet(screenID, 0, 0, 100, 100);

    display->line(0, 0, 100, 100);
    display->line(0, 100, 100, 0);

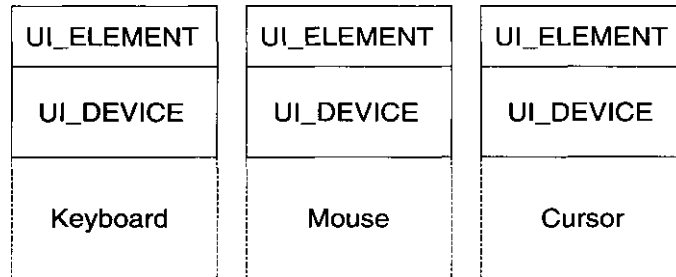
    // Copy the virtual buffer back to the screen.
    display->VirtualPut(screenID);
}
```





## CHAPTER 8 - UI ELEMENT

The UI\_ELEMENT class serves as the base class to all window object classes, all input device classes and several other specialized classes in OpenZinc Application Framework. The UI\_ELEMENT class works with the UI\_LIST class to form a doubly-linked list. Objects derived from UI\_ELEMENT are added to a list derived from UI\_LIST. This allows for the simple creation of a doubly-linked list containing any types of objects. Classes derived from the UI\_ELEMENT base class can be viewed in the following manner:



**NOTE:** In the figure above, the solid line denotes the base class (i.e., UI\_ELEMENT) and the dotted line shows the possible logical extensions of a derived class.

The UI\_ELEMENT class is declared in **UI\_GEN.HPP**. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UI_ELEMENT
{
public:
    virtual ~UI_ELEMENT(void);
    virtual ZIL_ICHAR *ClassName(void);
    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);
    int ListIndex(void);
    UI_ELEMENT *Next(void);
    UI_ELEMENT *Previous(void);

protected:
    UI_ELEMENT *previous, *next;

    UI_ELEMENT(void);
};
```

## General Members

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

- *previous* and *next* are pointers to the siblings of the element. If the sibling doesn't exist, the pointer will be NULL.

## UI\_ELEMENT::UI\_ELEMENT

### Syntax

```
#include <ui_gen.hpp>
```

```
UI_ELEMENT(void);
```

### 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 UI\_ELEMENT object. As the UI\_ELEMENT class cannot contain any information specific to the list being created, it is of little use by itself; rather, OpenZinc Application Framework uses the element class as a base class, relying on class members of derived classes for data storage and manipulation. This constructor is called when constructing one of these derived objects.

### Example 1

```
#include <ui_evt.hpp>
UI_DEVICE::UI_DEVICE (RAW_EVENT_type, ZIL_DEVICE_STATE_state) : UI_ELEMENT (),
    installed (FALSE), enabled (TRUE), type (_type), state (_state),
    display (NULL), eventManager (NULL)
{
    .
    .
    .
}
```

## Example 2

```
#include <ui_win.hpp>

UI_WINDOW_OBJECT::UI_WINDOW_OBJECT(int left, int top, int width, int height,
    WOF_FLAGS _woFlags, WOAF_FLAGS _woAdvancedFlags) : UI_ELEMENT(),
    woFlags(_woFlags), woAdvancedFlags(_woAdvancedFlags)
{
}
}
```

## UI\_ELEMENT::~~UI\_ELEMENT

### Syntax

```
#include <ui_gen.hpp>

virtual ~UI_ELEMENT(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 UI\_ELEMENT object. The destructor is declared virtual so that derived list element destructors can be called. (If the destructor for the UI\_ELEMENT class were not declared virtual, the programmer would need to call the destroy function associated with each derived class.)

### Example

```
#include <ui_gen.hpp>

ElementFunction()
{
    UI_ELEMENT element1;
    UI_ELEMENT *element2;
```

```
    // The element1 destructor is automatically called when the function ends,  
    delete element2;  
}
```

## **UI\_ELEMENT::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 function is a stub. Since all objects derived from UI\_ELEMENT use a virtual **ClassName()** function, this stub is necessary. If this function does get called, it simply returns NULL.

- *returnValue<sub>out</sub>* is NULL.

## **UI\_ELEMENT::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 is a stub. Since all objects derived from `UI_ELEMENT` use a virtual `Information()` function, this stub is necessary. If this function does get called, it simply returns `NULL`.

## UI\_ELEMENT::ListIndex

### Syntax

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

```
int ListIndex(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 ordinal position of the element in its parent `UI_LIST`.

## UI\_ELEMENT::Next

### Syntax

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

```
UI_ELEMENT *Next(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 next element, if one exists, in the list of elements.

- *returnValue<sub>out</sub>* is a pointer to the next element in the list. If there is not a next element, *returnValue* is NULL.

**NOTE:** The **Next()** function is also used by window objects and input devices. In each case, the function is overloaded to return an object pointer typecast according to the context. For example, window objects generally return a **UI\_WINDOW\_OBJECT** pointer when **Next()** is called:

```
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")
+ new UIW_TEXT(1, 1, 10, 5, "Hello world.", 256);

for (UI_WINDOW_OBJECT *object = window->First(); object;
     object = object->Next())
```

Input devices, however, return a **UI\_DEVICE** pointer:

```
UI_DISPLAY *display = new UI_TEXT_DISPLAY;
UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
*eventManager
+ new UID_KEYBOARD
+ new UID_MOUSE
+ new UID_CURSOR;

for (UI_DEVICE *device = eventManager->First(); device;
     device = device->Next());
```

Some other class objects return specific element pointers (e.g., **UI\_QUEUE\_ELEMENT**, **UI\_REGION\_ELEMENT**, **UIW\_POP\_UP\_ITEM**). Refer to each class definition for information about the return value of **Next()**.

## UIELEMENT::Previous

### Syntax

```
#include <ui_win.hpp>

UIELEMENT *Previous(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 previous element, if one exists, in the list of elements.

- *returnValue<sub>out</sub>* is a pointer to the previous element in the list. If there is not a previous element, *returnValue* is NULL.

**NOTE:** The **Previous()** function is also used by window objects and input devices. In each case, the function is overloaded to return an object pointer typecast according to the context. For example, window objects generally return a **UI\_WINDOW\_OBJECT** pointer when **Previous()** is called:

```
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")
+ new UIW_TEXT(1, 1, 10, 5, "Hello world.", 256);

for (UI_WINDOW_OBJECT *object = window->Last(); object;
object = object->Previous())
```

Input devices, however, return a **UI\_DEVICE** pointer:

```
UI_DISPLAY *display = new UI_TEXT_DISPLAY;
UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
```



```
*eventManager
+ new UID_KEYBOARD
+ new UID_MOUSE
+ new UID_CURSOR;

for (UI_DEVICE *device = eventManager->Last(); device;
     device = device->Previous());
```

Some other class objects return specific element pointers (e.g., `UI_QUEUE_ELEMENT`, `UI_REGION_ELEMENT`, `UIW_POP_UP_ITEM`). Refer to each class definition for information about the return value of **Previous()**.

## CHAPTER 9 - UI ERROR STUB

The `UI_ERROR_STUB` class is the base class for the error system. The error system is used to display an error message and to get a response from the end-user. The `UI_ERROR_STUB` class defines the functionality that must exist in the error system. It is an abstract class, so only classes derived from `UI_ERROR_STUB`, such as `UI_ERROR_SYSTEM`, can be created.

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

```
class ZIL_EXPORT_CLASS UI_ERROR_STUB : public ZIL_INTERNATIONAL
{
public:
    virtual ~UI_ERROR_STUB(void);
    static void Beep(void);
    UIS_STATUS ReportError(UI_WINDOW_MANAGER *windowManager,
        UIS_STATUS errorStatus, ZIL_ICHAR *format, ...);
    UIS_STATUS ReportError(UI_WINDOW_MANAGER *windowManager,
        ZIL_ICHAR *titleMessage, UIS_STATUS errorStatus, ZIL_ICHAR *format,
        ...);
    virtual UIS_STATUS ErrorMessage(UI_WINDOW_MANAGER *windowManager,
        UIS_STATUS errorStatus, ZIL_ICHAR *message,
        ZIL_ICHAR *titleMessage = ZIL_NULLP(ZIL_ICHAR)) = 0;
};
```

### General Members

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

### UI\_ERROR\_STUB::~~UI\_ERROR\_STUB

#### Syntax

```
#include <ui_win.hpp>

virtual ~UI_ERROR_STUB(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 `UI_ERROR_STUB` object.

## UI\_ERROR\_STUB::Beep

### Syntax

```
#include <ui_win.hpp>

static void Beep(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 produces a beep.

## UI\_ERROR\_STUB::ErrorMessage

### Syntax

```
#include <ui_win.hpp>

virtual UIS_STATUS ErrorMessage(UI_WINDOW_MANAGER *windowManager,
    UIS_STATUS errorStatus, ZIL_ICHAR *message,
```

```
ZIL_ICHAR *titleMessage = 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 pure virtual function displays the error window. See "Chapter 10—UI\_ERROR\_SYSTEM" for details on the error system's implementation of this function.

**NOTE:** This function does not accept a variable length argument format. If **printf** style formatting is required, use the **ReportError()** function.

- *returnValue<sub>out</sub>* identifies the user's action on the error window. *returnValue* is WOSJNVALID if the "OK" button is pressed or WOS\_NO\_STATUS if the "Cancel" button is pressed.
- *windowManager<sub>in</sub>* is a pointer to the Window Manager.
- *errorStatus<sub>in</sub>* specifies what error window button options to present to the end-user. *errorStatus* can be set to one of the following:

**WOS\_NVALID**—If this status is set, the error window will contain "OK" and "CANCEL" buttons. Selecting "OK" causes the error window to be deleted and the field's value to be restored to the value it contained before the invalid entry was made. Pressing the "CANCEL" button causes the error window to be deleted and the invalid field entry to remain.

**WOS\_NO\_STATUS**—If this status is set, the error window will contain only an "OK" button. Pressing the "OK" button causes the error window to be deleted and the field's value to be restored to the value it contained before the invalid entry was made.

- *message<sub>in</sub>* is the message to be displayed on the window.
- *titleMessage<sub>in</sub>* is the string to be displayed in the error window's title bar.

## UI\_ERROR\_STUB::ReportError

### Syntax

```
#include <ui_win.hpp>

UIS_STATUS ReportError(UI_WINDOW_MANAGER *windowManager,
    UIS_STATUS errorStatus, ZIL_ICHAR *format, ...);
    or
UIS_STATUS ReportError(UI_WINDOW_MANAGER *windowManager,
    ZIL_ICHAR *titleMessage, UIS_STATUS errorStatus, 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

These overloaded functions display the error window using a variable-length argument list. These functions create a string from the argument list and call **ErrorMessage()**.

The first overloaded function can be used to specify the button options and the message.

- *returnValue<sub>out</sub>* identifies the user's action on the error window. For more details, see the description of *returnValue* with the **ErrorMessage( )** function description.
- *windowManager<sub>in</sub>* is a pointer to the Window Manager.
- *errorStatus<sub>in</sub>* specifies what error window button options to present to the end-user. For more details, see the description of *errorStatus* with the **ErrorMessage()** function description.
- *format<sub>in</sub>* is the **printf** style format string that specifies how the error string is to be displayed.
- *...<sub>in</sub>* is the variable-length argument list that contains any arguments required by *format*.

The second overloaded function can be used to specify the button options, the error message and the title bar string.

- *returnValue*<sub>out</sub> identifies the user's action on the error window. For more details, see the description of *returnValue* with the **ErrorMessage()** function description.
- *windowManager*<sub>in</sub> is a pointer to the Window Manager.
- *titleMessage*<sub>in</sub> is the string to be displayed in the error window's title bar.
- *errorStatus*<sub>in</sub> specifies what error window button options to present to the end-user. For more details, see the description of *errorStatus* with the **ErrorMessage()** function description.
- *format*<sub>in</sub> is the **printf** style format string that specifies how the string is to be displayed.
- *...*<sub>in</sub> is the variable-length argument list that contains any arguments required by *format*.



## CHAPTER 10 - UI ERROR SYSTEM

The `UI_ERROR_SYSTEM` class is used to report run-time errors. It displays an error window with one or more buttons allowing the end-user to specify what action to take. The programmer provides a message to be displayed in the error window as well as the title for the error window, if desired. If the environment where the application is running (e.g., Windows) has a native error system, then `UI_ERROR_SYSTEM` calls that error system.

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

```
class ZIL_EXPORT_CLASS UI_ERROR_SYSTEM : public UI_ERROR_STUB
{
public:
    static ZIL_ICHAR _className[];
    static int defaultInitialized;

    UI_ERROR_SYSTEM(void);
    virtual ~UI_ERROR_SYSTEM(void);
    virtual UIS_STATUS ErrorMessage(Ul_WINDOW_MANAGER *windowManager,
        UIS_STATUS errorStatus, ZIL_ICHAR *message,
        ZIL_ICHAR *titleMessage == ZIL_NULLP (ZIL_ICHAR) );

    void SetLanguage(const ZIL_ICHAR *languageName);

protected:
    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 `UI_ERROR_SYSTEM` class, `_className` is "UI\_ERROR\_SYSTEM."
- `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. `defaultInitialized` is set to TRUE when the strings are set up in the object's constructor.
- `myLanguage` is the `ZIL_LANGUAGE` object that contains the string translations for this object.



## **UI\_ERROR\_SYSTEM::UI\_ERROR\_SYSTEM**

### **Syntax**

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

```
UI_ERROR_SYSTEM(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 UI\_ERROR\_SYSTEM class object.

### **Example**

```
#include <ui_win.hpp>

main ()

{

    // Install the error system.
    UI_ERROR_SYSTEM *errorSystem = new UI_ERROR_SYSTEM();
    UI_WINDOW_OBJECT::errorSystem = errorSystem;

    // Clean up.
    delete errorSystem;
    delete windowManager;
    delete eventManager;
    delete display;
}
```

## UI\_ERROR\_SYSTEM::~~UI\_ERROR\_SYSTEM

### Syntax

```
#include <ui_win.hpp>

virtual ~UI_ERROR_SYSTEM(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 UI\_ERROR\_SYSTEM object.

### Example

```
#include <ui_win.hpp>

main()
{

    // Install the error system.
    UI_ERROR_SYSTEM *errorSystem = new UI_ERROR_SYSTEM();
    UI_WINDOW_OBJECT::errorSystem = errorSystem;

    // Clean up.
    delete errorSystem;
    delete windowManager;
    delete eventManager;
    delete display;
```

## UI\_ERROR\_SYSTEM::ErrorMessage

### Syntax

```
#include <ui_win.hpp>

virtual UIS_STATUS ErrorMessage(UI_WINDOW_MANAGER *windowManager,
    UIS_STATUS errorStatus, ZIL_ICHAR *message,
    ZIL_ICHAR *titleMessage = ZIL_NULLP(ZIL_ICHAR));
```

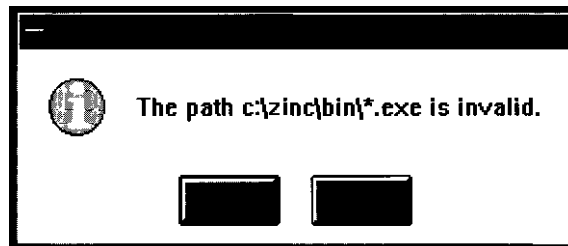
### Portability

This function is available on the following environments:

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

### Remarks

This virtual function beeps and displays an error window. The programmer can specify a message and a title to appear on the window. If no title is specified, an appropriate language-specific error title will be displayed. This function is declared virtual so that any derived error system class can override its default operation. The figure below shows a graphical UI\_ERROR\_SYSTEM presentation window:



**NOTE:** This function does not accept a variable length argument format. If printf-type formatting is required, use the **UI\_ERROR\_STUB::ReportError()** function.

- *returnValue<sub>out</sub>* identifies the user's action on the error window. *returnValue* is **WOS\_INVALID** if the "OK" button is pressed or **WOS\_NO\_STATUS** if the "Cancel" button is pressed.

- *windowManager<sub>in</sub>* is a pointer to the Window Manager.
- *errorStatus<sub>in</sub>* specifies what error window button options to present to the end-user. *errorStatus* can be set to one of the following:

**WOS\_INVALID**—If this status is set, the error window will contain "OK" and "CANCEL" buttons. Selecting "OK" causes the error window to be deleted and the field's value to be restored to the value it contained before the invalid entry was made. Pressing the "CANCEL" button causes the error window to be deleted and the invalid field entry to remain.

**WOS\_NO\_STATUS**—If this status is set, the error window will contain only an "OK" button. Pressing the "OK" button causes the error window to be deleted and the field's value to be restored to the value it contained before the invalid entry was made.

- *message<sub>in</sub>* is the message to be displayed on the window.
- *titleMessage<sub>in</sub>* is the string to be displayed in the error window's title bar.

**NOTE:** In DOS mode the **ErrorMessage( )** function looks for an icon called "ASTERISK." This icon must be in the *UI\_WINDOW\_OBJECT::defaultStorage* .DAT file in order for the error icon to be displayed. This icon is placed in any .DAT file created by a OpenZinc utility, including the Designer.

## Example

```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_OBJECT *item)
{
    item->errorSystem->ErrorMessage(item->windowManager, WOS_INVALID,
        "The path c: \\zinc\\bin\\*.exe is invalid.", "Invalid Path");
}
}
```

## UI\_ERROR\_SYSTEM::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 ZINC\SOURCE\INTL directory to the \ZINC\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<sub>in</sub>* is the two-letter ISO language code identifying which language the object should use.

## CHAPTER 11 - UI EVENT

The UI\_EVENT structure manages all information pertaining to an event. The UI\_EVENT structure is used to pass events through OpenZinc Application Framework (from the Event Manager to the Window Manager).

The UI\_EVENT structure is declared in UI\_EVT.HPP. Its public and protected members are:

```
struct ZIL_EXPORT_CLASS UI_EVENT
{
    // Declaration of classes used by UI_EVENT.
    friend class ZIL_EXPORT_CLASS UI_DEVICE;
    friend class ZIL_EXPORT_CLASS UI_DISPLAY;
    friend class ZIL_EXPORT_CLASS UI_EVENT_MANAGER;
    friend class ZIL_EXPORT_CLASS UI_WINDOW_OBJECT;
    friend class ZIL_EXPORT_CLASS UIW_WINDOW;
    friend class ZIL_EXPORT_CLASS UI_WINDOW_MANAGER;

    EVENT_TYPE type;
    ZIL_RAW_CODE rawCode;
    ZIL_RAW_CODE modifiers;
    #if defined(ZIL_MSWINDOWS)
        MSG message;
    #elif defined(ZIL_OS2)
        QMSG message;
    #elif defined(ZIL_MOTIF)
        XEvent message;
    #elif defined(ZIL_MACINTOSH)
        EventRecord message;
    #elif defined(ZIL_NEXTSTEP)
        NXEvent message;
    #endif

    union
    {
        UI_KEY key;
        UI_REGION region;
        UI_POSITION position;
        UI_SCROLL_INFORMATION scroll;
        UI_EVENT *event;

        UI_DEVICE *device;
        UI_DISPLAY *display;
        UI_EVENT_MANAGER *eventManager;
        UI_WINDOW_OBJECT *windowObject;
        UIW_WINDOW *window;
        UI_WINDOW_MANAGER *windowManager;

        void *data;
    };

    UI_EVENT(void);
    UI_EVENT(EVENT_TYPE type, ZIL_RAW_CODE rawCode = 0);
    UI_EVENT(EVENT_TYPE type, ZIL_RAW_CODE rawCode, const UI_KEY &key);
    UI_EVENT(EVENT_TYPE type, ZIL_RAW_CODE rawCode,
              const UI_REGION &region);
    UI_EVENT(EVENT_TYPE type, ZIL_RAW_CODE rawCode,
              const UI_POSITION &position);
    UI_EVENT(EVENT_TYPE type, ZIL_RAW_CODE rawCode,
              const UI_SCROLL_INFORMATION &scroll);
    #if defined(ZIL_MSWINDOWS)
```

```

    UI_EVENT(EVENT_TYPE type, HWND hWnd, UINT wParam, WPARAM wParam,
             LPARAM lParam);
#elif defined(ZIL_OS2)
    UI_EVENT(EVENT_TYPE type, HWND hWnd, ULONG msg, MPARAM mp1, MPARAM mp2);
#elif defined(ZIL_MOTIF)
    UI_EVENT(EVENT_TYPE type, XEvent kxevent);
#elif defined(ZIL_MACINTOSH)
    UI_EVENT(EVENT_TYPE type, EventRecord kmevent);
#elif defined(ZIL_NEXTSTEP)
    UI_EVENT(EVENT_TYPE type, NXEvent &nevent);
#endif
EVENT_TYPE InputType(void) const;

```

## General Members

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

- *type* is the type of event. Events are numbered as follows:

**-32,767 to -1,000**—Reserved by OpenZinc Application Framework for future use.

**-999 to -1**—Reserved by OpenZinc Application Framework for system messages. System messages are declared in **UI\_EVT.HPP**. A full description of these messages is given in "Appendix B—System Events" of *Programmer's Reference Volume 2*.

**0 to 99**—Reserved for raw device identifications. The following constants (declared in **UI\_EVT.HPP**) are pre-defined:

**E\_CURSOR(50)**—Identification for the UID\_CURSOR class.

**E\_DEVICE(99)**—Identification used to define a generic device.

**E\_KEY(10)**—Identification for the UID\_KEYBOARD class.

**E\_MACINTOSH(4)**—Identification for Macintosh events.

**E\_MOTIF(3)**—Identification for Motif events.

**E\_MOUSE(30)**—Identification for the UID\_MOUSE class.

**E\_MSWINDOWS(1)**—Identification for MS Windows events.

**E\_NEXTSTEP(11)**—Identification for NEXTSTEP events.

E\_OS2(2)—Identification for OS/2 events.

The following additional raw device identifications are reserved by OpenZinc Application Framework for future use: 12-19, 31-39, 51-59, 70-79 and 90-98. The remaining values 5-9, 20-29, 40-49, 60-69 and 80-89 can be used by the programmer.

**100 to 9,999**—Reserved by OpenZinc Application Framework for logical events. Logical messages are declared in **UI\_EVT.HPP**. A full description of these messages is given in "Appendix C—Logical Events" of *Programmer's Reference Volume 2*.

**10,000 to 32,767**—Available to the programmer for private use. These values are not used by OpenZinc Application Framework.

- *rawCode* is the raw code value associated with the event. The following devices (declared in **UI\_EVT.HPP**) use the *rawCode* event field:

**UID\_KEYBOARD**—The *rawCode* for the keyboard device is the raw scan code associated with the key. For example, pressing <F1> in DOS generates a raw scan code of 0x3B00. In this case, the UI\_EVENT structure would contain the following values:

```
event.type = E_KEY;  
event.rawCode = 0x3B00;  
event.key.value = 0;           // low 8 bits of rawCode  
event.key.shiftState = 0;
```

**NOTE:** Curses does not place scan code values in *rawCode*. Instead, *rawCode* will contain the key's ASCII value.

**UID\_MOUSE**—The *rawCode* for the mouse device is the keyboard shift state (low 8 bits) and the mouse button state (high 8 bits). For example, pressing the left mouse button while holding the <Left-shift> key generates a raw code of 0x0102 (0x0002 for the <Left-shift> key and 0x0100 for the left-mouse button). In this case, the UI\_EVENT structure would contain the following values:

```
event.type = E_MOUSE;  
event.rawCode = 0x0102;  
event.position.column = <current mouse column position>;  
event.position.line = <current mouse row position>;
```



- *modifiers* is a bit field that indicates which modifier keys (i.e., shift keys, meta keys, etc.) were pressed at the time the event occurred.
- *message* is the message received from the graphical operating system if the application is running in such an environment.
- *key*, *region*, *position*, *scroll* and *data* are types of specific information associated with the event.
- *event*, *device*, *display*, *eventManager*, *windowObject*, *window* and *windowManager* are used for routing events.

## UI\_EVENT::UI\_EVENT

### Syntax

```

UI_EVENT(void);
    or
UI_EVENT(EVENT_TYPE type, ZIL_RAW_CODE rawCode = 0);
    or
UI_EVENT(EVENT_TYPE type, ZIL_RAW_CODE rawCode, const UI_KEY &key)\
    or
UI_EVENT(EVENT_TYPE type, ZIL_RAW_CODE rawCode,
    const UI_REGION &region);
    or
UI_EVENT(EVENT_TYPE type, ZIL_RAW_CODE rawCode,
    const UI_POSITION &position);
    or
UI_EVENT(EVENT_TYPE type, ZIL_RAW_CODE rawCode,
    const UI_SCROLL_INFORMATION &scroll),
    or
UI_EVENT(EVENT_TYPE type, HWND hWnd, UINT wParam, WPARAM wParam,
    LPARAM lParam)\
    or
UI_EVENT(EVENT_TYPE type, HWND hWnd, ULONG msg, MPARAM mp1,
    MPARAM mp2);
    or
UI_EVENT(EVENT_TYPE _type, XEvent &event);
    or
UI_EVENT(EVENT_TYPE type, EventRecord &event)\

```

*or*  
UI\_EVENT(EVENT\_TYPE *type*, NXEvent *&nevent*);

## Portability

This function is available on the following environments:

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

## Remarks

The first overloaded constructor takes no arguments. It creates a basic event structure with no special initialization.

The second overloaded constructor initializes an event structure with the following arguments:

- *type<sub>in</sub>* contains a valid EVENT\_TYPE.
- *rawCode<sub>in</sub>* contains a valid ZIL\_RAW\_CODE.

The third overloaded constructor initializes an event structure with the following arguments:

- *type<sub>in</sub>* contains a valid EVENTTYPE.
- *rawCode<sub>in</sub>* contains a valid ZIL\_RAW\_CODE.
- *key<sub>in</sub>* contains the address of a UI\_KEY structure.

The fourth overloaded constructor initializes an event structure with the following arguments:

- *type<sub>in</sub>* contains a valid EVENT\_TYPE.
- *rawCode<sub>in</sub>* contains a valid ZIL\_RAW\_CODE.
- *region<sub>in</sub>* contains the address of a UI\_REGION structure.

The fifth overloaded constructor initializes an event structure with the following

arguments:

- *type<sub>in</sub>* contains a valid EVENT\_TYPE.
- *rawCode<sub>in</sub>* contains a valid ZIL\_RAW\_CODE.
- *position<sub>in</sub>* contains the address of a UI\_POSITION structure.

The sixth overloaded constructor initializes an event structure with the following arguments:

- *type<sub>in</sub>* contains a valid EVENT\_TYPE.
- *rawCode<sub>in</sub>* contains a valid ZIL\_RAW\_CODE.
- *scroll<sub>in</sub>* contains the address of a UI\_SCROLL\_INFORMATION structure.

The seventh overloaded constructor is used only for Windows and Windows NT programs. It initializes an event structure with the following arguments:

- *type<sub>in</sub>* contains a valid EVENT\_TYPE.
- *hWnd<sub>in</sub>* contains a handle to a window.
- *wMsg<sub>in</sub>* contains an input message.
- *wParam<sub>in</sub>* is a UINT value containing specific message information.
- *lParam<sub>in</sub>* is an LPARAM value containing specific message information.

The eighth overloaded constructor is used only for OS/2 programs. It initializes an event structure with the following arguments:

- *type<sub>in</sub>* contains a valid EVENT\_TYPE.
- *hWnd<sub>in</sub>* contains a handle to a window.
- *msg<sub>in</sub>* contains an input message.
- *mpl<sub>in</sub>* is an MPARAM value containing specific message information.
- *mp2<sub>in</sub>* is an MPARAM value containing specific message information.

The ninth overloaded constructor is used only for Motif programs. It initializes an event structure with the following arguments:

- *\_type<sub>in</sub>* contains a valid EVENT\_TYPE.
- *xevent<sub>in</sub>* is an XEvent value containing specific message information.

The tenth overloaded constructor is used only for Macintosh programs. It initializes an event structure with the following arguments:

- *type<sub>in</sub>* contains a valid EVENT\_TYPE.
- *mevent<sub>in</sub>* is a Macintosh EventRecord value containing specific message information.

The eleventh overloaded constructor is used only for NEXTSTEP programs. It initializes an event structure with the following arguments:

- *type<sub>in</sub>* contains a valid EVENT\_TYPE.
- *nevent<sub>in</sub>* is a NEXTSTEP NXEvent value containing specific message information.

### Example 1

```
#include <ui_win.hpp>
main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_TEXT_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;
    UI_WINDOW_MANAGER *windowManager = new UI_WINDOW_MANAGER(display,
        eventManager);

    EVENT_TYPE ccode;

    do
    {
        // Get an event from the event manager.
        UI_EVENT event;
        eventManager->Get(event, Q_NORMAL);

        // Pass the event to the window manager.
        windowManager->Event(event);
    } while (ccode != L_EXIT);
```

```

}

static void Exit(UI_WINDOW_OBJECT *item, UI_EVENT &event, EVENT_TYPE ccode)
{
    // Send an L_EXIT message through the system,
    event.type = L_EXIT;
    UI_EVENT_MANAGER *eventManager = ((UIW_POP_UP_ITEM *)item)->eventManager;
    eventManager->Put(event, Q_BEGIN);
}

```

## Example 2

```

void UI_WINDOW_MANAGER::Add(UI_WINDOW_OBJECT *object)

{

    // Flag the old object as non-current,
    if (firstObject && object != firstObject)
        firstObject->Event(UI_EVENT(S_NON_CURRENT));

    // Flag the new object as current.
    if (object != firstObject && UI_LIST::Index(object) != -1)
    {
        UI_LIST::Subtract(object);
        UI_LIST::Add(firstObject, object);
    }
    else if (object != firstObject)
    {
        UI_LIST::Add(firstObject, object);
        object->Event(UI_EVENT(S_INITIALIZE));
        object->Event(UI_EVENT(S_CREATE));
    }
    display->RegionDefine(object->screenID, object->>trueRegion);

}

```

## ULEVENT::InputType

### Syntax

```

#include <ui_evt.hpp>

EVENT_TYPE InputType(void) const;

```

## Portability

This function is available on the following environments:

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

## Remarks

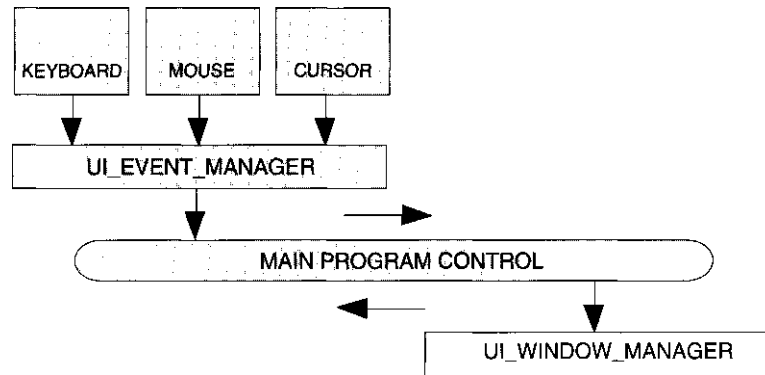
This function returns the type of device that generated the event if the event was generated by a mouse or keyboard in a graphical operating system.

- *returnValue<sub>out</sub>* indicates the type of device that generated the event. *returnValue* will be either E\_MOUSE if the event was a mouse event, or E\_KEY if the event was a keyboard event.



## CHAPTER 12 - UI EVENT MANAGER

The UI\_EVENT\_MANAGER class manages input devices and the event queue that temporarily stores messages waiting to be processed. The graphic illustration below shows the conceptual operation of the Event Manager within the library:



The controlling portion of the UI\_EVENT\_MANAGER class contains a list of input devices. Whenever an event is requested, the Event Manager polls each device, allowing it to place any events it may have on the event queue. This portion of the UI\_EVENT\_MANAGER class is used only in those operating systems that do not provide an event driven messaging system (e.g., DOS and Curses).

The storage portion of the UI\_EVENT\_MANAGER class is implemented as an array of UI\_EVENT structures. The size of this array is specified by the programmer when the Event Manager class is constructed. Input devices place events on the event queue so that they may be processed by the system. In addition to input devices, the operating system (only those operating systems that provide an event driven messaging system) and the programmer may also place events on the queue.

The UI\_EVENT\_MANAGER class is declared in **UI\_EVT.HPP**. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UI_EVENT_MANAGER : public UI_LIST
public:
    UI_EVENT_MANAGER(UI_DISPLAY *display, int noOfElements = 100);
    virtual ~UI_EVENT_MANAGER(void);
    EVENT_TYPE DevicePosition(ZIL_DEVICE_TYPE deviceType, int column,
        int line);
```



```

EVENT_TYPE DeviceState(ZIL_DEVICE_TYPE deviceType,
    ZIL_DEVICE_STATE deviceState);
EVENT_TYPE DeviceImage(ZIL_DEVICE_TYPE deviceType,
    DEVICE_IMAGE deviceImage);
virtual EVENT_TYPE Event(const UI_EVENT &event,
    ZIL_DEVICE_TYPE deviceType = E_DEVICE);
virtual int Get(UI_EVENT &event, Q_FLAGS flags = Q_NORMAL);
virtual void Put(const UI_EVENT &event, Q_FLAGS flags = Q_END);
Q_FLAGS QFlags(void);

// List members.
void Add(UI_DEVICE *device);
UI_DEVICE *Current(void);
UI_DEVICE *First(void);
UI_DEVICE *Last(void);
void Subtract(UI_DEVICE *device);
UI_EVENT_MANAGER &operator+(UI_DEVICE *device);
UI_EVENT_MANAGER &operator-(UI_DEVICE *device);

// Version 2.0 and 1.0 compatibility.
UI_EVENT_MANAGER(int noOfElements, UI_DISPLAY *display);

protected:
    int level;
    Q_FLAGS qFlags;
    UI_DISPLAY *display;
    UI_QUEUE_BLOCK queueBlock;
#ifdef defined(ZIL_OS2)
    HMQ hmq;
#endif

```

## General Members

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

- *level* indicates if a recursive call to the **Get()** function is being made. If *level* is 1, then only a first-level call has been made to **Get()**. *level* is incremented each time **Get()** is called, and decremented each time it exits. Thus, if *level* is greater than 1, the function is at a recursive level.
- *qFlags* contains the flag settings for the current first-level call to **Get()**.
- *display* is a pointer to the current display class.
- *queueBlock* is a pointer to the event queue, which contains all of the unprocessed messages sent by the devices, the operating system, or the programmer.
- *hmq* is a pointer to the OS/2 message queue.

## UI\_EVENT\_MANAGER::UI\_EVENT\_MANAGER

### Syntax

```
#include <ui_evt.hpp>
```

```
UI_EVENT_MANAGER(UI_DISPLAY *display, int noOfElements = 100);
```

### 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 `UI_EVENT_MANAGER` class object. It must be called after the display class constructor has been called.

- *display<sub>in</sub>* is a pointer to the screen display. This pointer is used by input devices when they display their information to the screen display (e.g., the blinking cursor of the `UID_CURSOR` class object).
- *noOfElements<sub>in</sub>* tells the maximum number of elements to reserve in the event queue. The Event Manager automatically allocates space for *noOfElements*.

### Example

```
#include <ui_win.hpp>

main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_TEXT_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display, 100);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;
    UI_WINDOW_MANAGER *windowManager = new UI_WINDOW_MANAGER(display,
        eventManager);

    // Restore the system. We must explicitly call the destructor for the
    // window manager and event manager to preserve the creation order.
}
```

```

delete windowManager;
delete eventManager;
delete display;
return (0);

```

## UI\_EVENT\_MANAGER::~~UI\_EVENT\_MANAGER

### Syntax

```

#include <ui_evt.hpp>

virtual ~UI_EVENT_MANAGER(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 UI\_EVENT\_MANAGER object and destroys the class information of any input device that remains attached to the Event Manager.

### Example

```

#include <ui_win.hpp>

main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_TEXT_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;
    UI_WINDOW_MANAGER *windowManager = new UI_WINDOW_MANAGER(display,
        eventManager);

    // Restore the system. We must explicitly call the destructor for the
    // window manager and event manager to preserve the creation order,
    delete windowManager;
}

```

```
delete eventManager;
delete display;
return (0) ;
```

## UI\_EVENT\_MANAGER::Add

### Syntax

```
#include <ui_evt.hpp>

virtual void Add(UI_DEVICE *device);
```

### 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 device to the Event Manager. When devices are added to the Event Manager, they are ordered so that devices with a higher priority (i.e., lower device type value) will be at the beginning of the Event Manager's list.

- *device<sub>in</sub>* is the device to be added to the Event Manager.

## UI\_EVENT\_MANAGER::Current

### Syntax

```
#include <ui_evt.hpp>

UI_DEVICE *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 device in the Event Manager's list.

- *returnValue<sub>out</sub>* is the current device in the Event Manager's list.

## UI\_EVENT\_MANAGER::DeviceImage

### Syntax

```
#include <ui_evt.hpp>
```

```
EVENT_TYPE DeviceImage(ZIL_DEVICE_TYPE deviceType,  
DEVICE_IMAGE deviceImage);
```

## 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 image displayed by the input device specified by *deviceType*.

- *returnValue<sub>out</sub>* is the new state of the device (i.e., its image as specified in *deviceImage*).
- *deviceType<sub>in</sub>* is the device identification where the image message is to be sent. The following device types (declared in **UI\_EVT.HPP**) may be specified:

**E\_CURSOR**—Sends the state information to the `UID_CURSOR` class object (if it is in the device list).

**E\_DEVICE**—Sends the state information to an input device whose device type is `E_DEVICE` (if it is in the device list).

**E\_MOUSE**—Sends the state information to the `UID_MOUSE` class object (if it is in the device list.)

- *devicelimage<sub>n</sub>* is the new image of the device. For mouse, pen, and cursor devices, the allowable image changes (declared in **UI\_EVT.HPP**) are:

**E\_CURSOR**—The `UID_CURSOR` class recognizes the following image information:

**DC\_INSERT**—Changes the cursor to an insert cursor. In DOS graphics mode, if *devicelimage* is `DC_INSERT`, the cursor device displays a thick vertical bar cursor on the screen. In DOS text mode, the `DC_INSERT` cursor is a wide box. This image applies to DOS only.

**DC\_OVERSTRIKE**—Changes the cursor to an overstrike cursor. In DOS graphics mode, if *devicelimage* is `DC_OVERSTRIKE`, the cursor device displays a thin vertical bar cursor on the screen. In DOS text mode, the `DC_OVERSTRIKE` cursor is a short, wide underline. This image applies to DOS only, (e.g., a thin vertical bar).

**E\_MOUSE**—The `UID_MOUSE` class recognizes the following image information:

**DM\_DIAGONAL\_ULLR**—Displays the image shown when sizing the top-left or bottom-right corner of a window.

**DM\_DIAGONAL\_LLUR**—Displays the image shown when sizing the top-right or bottom-left corner of a window.

**DM\_EDIT**—Displays the image shown when positioned over an editable field.

**DM\_HORIZONTAL**—Displays the image shown when sizing a window horizontally.

**DM\_MOVE**—Displays the image shown when indicating that the object is to be moved.

**DM\_POSITION\_IMAGE**—Displays the image shown when indicating that something is to be positioned by the device.

**DM\_VERTICAL**—Displays the image shown when sizing a window vertically.

**DM\_VIEW**—Displays the default image, typically an arrow.

**DM\_WAIT**—Displays the image shown to indicate to the user that some processing is taking place and that he should wait.

**NOTE:** Because OpenZinc allows the graphical operating systems to handle their images, not all of these images may be supported in all environments.

## Example

```
#include <ui_evt.hpp>

main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_GRAPHICS_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;

    // Display an hour glass until the program is ready to receive user input.
    eventManager->DeviceImage(E_MOUSE, DM_WAIT);

}
```

## UI\_EVENT\_MANAGER::DevicePosition

### Syntax

```
#include <ui_evt.hpp>

void DevicePosition(ZIL_DEVICE_TYPE deviceType, int column, int line);
```

## 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 position for the device (e.g., mouse, pen, or cursor). Some graphical operating systems may not allow this type of action for all devices.

- *deviceType<sub>in</sub>* is the type of device (e.g., E\_CURSOR, or E\_MOUSE) for which the message is intended.
- *column<sub>in</sub>* and *line<sub>in</sub>* is the position to where the device will be moved. The value of *column* and *line* depends on the type of display mode in which the application is running. For example, if the cursor is to be positioned at the center of the screen while the application is running in text mode (i.e., an 80 column by 25 line screen) the position values should be:

```
column = 40;
line = 13;
eventManager->DevicePosition(E_CURSOR, column, line);
```

If on the other hand, the application is running in a 640 column by 480 line graphics mode, the position values should be:

```
column = 320;
line = 240;
eventManager->DevicePosition(E_CURSOR, column, line);
```

## Example

```
#include <ui_evt.hpp>

main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_GRAPHICS_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;

    // Reposition the cursor at the top-left side of the screen.
    eventManager->DevicePosition(E_CURSOR, 0, 0);
}
```



```
}
```

## **UI EVENT MANAGER::DeviceState**

### **Syntax**

```
#include <ui_evt.hpp>
```

```
EVENT_TYPE DeviceState(ZIL_DEVICE_TYPE deviceType,  
ZIL_DEVICE_STATE deviceState);
```

### **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 state for the input device specified by *deviceType*.

- *returnValue<sub>out</sub>* is the new state of the device.
- *deviceType<sub>in</sub>* is the device identification where the state message is to be sent. The following device types (declared in **UI\_EVT.HPP**) may be specified:

**E\_CURSOR**—Sends the state information to the UID\_CURSOR class object (if it is in the device list).

**E\_DEVICE**—Sends the state information to an input device whose device type is E\_DEVICE (if it is in the device list).

**E\_MOUSE**—Sends the state information to the UID\_MOUSE class object (if it is in the device list.)

**E\_KEY**—Sends the state information to the UID\_KEYBOARD class object (if it is in the device list).

- *deviceState<sub>in</sub>* is the new state of the device. Allowable state changes (declared in `UI_EVT.HPP`) are:

**D\_HIDE**—Hides the specified device's image. If *deviceType* is `E_DEVICE`, all devices in the Event Manager's device list are sent the `D_HIDE` message.

**D\_ON**—Turns the specified device on. If *deviceType* is `E_DEVICE`, all devices in the Event Manager's device list are sent the `D_ON` message.

**D\_OFF**—Turns the specified device off. If *deviceType* is `E_DEVICE`, all devices in the Event Manager's device list are sent the `D_OFF` message.

**D\_STATE**—Gets the state information associated with the specified device. If *deviceType* is `E_DEVICE`, only the state of the last device in the Event Manager's device list is returned.

**Other device states**—These must be recognized by the device whose type is *deviceType*. For example, the `UID_MOUSE` class also recognizes the following state information:

**DM\_DIAGONAL\_ULLR**—Displays the image shown when sizing the top-left or bottom-right corner of a window.

**DM\_DIAGONAL\_LLUR**—Displays the image shown when sizing the top-right or bottom-left corner of a window.

**DM\_EDIT**—Displays the image shown when positioned over an editable field.

**DM\_HORIZONTAL**—Displays the image shown when sizing a window horizontally.

**DM\_MOVE**—Displays the image shown when indicating that the object is to be moved.

**DM\_POSITION\_IMAGE**—Displays the image shown when indicating that something is to be positioned by the device.

**DM\_VERTICAL**—Displays the image shown when sizing a window vertically.

**DM\_VIEW**—Displays the default image, typically an arrow.

DM\_WAIT—Displays the image shown to indicate to the user that some processing is taking place and that he should wait.

## Example

```
#include <ui_evt.hpp>

main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_GRAPHICS_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;

    // Display an hour glass until the program is ready to receive user input.
    eventManager->DeviceState(E_MOUSE, DM_WAIT);

}
```

## UI\_EVENT\_MANAGER::Event

### Syntax

```
#include <ui_evt.hpp>

virtual EVENT_TYPE Event(const UI_EVENT &event,
    ZIL_DEVICE_TYPE deviceType = E_DEVICE);
```

### 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 allows the programmer to communicate with devices through the Event Manager. This permits the programmer to change the interaction of input devices without having a pointer to the device.

- *event<sub>in</sub>* is the message to be passed to an input device.
- *deviceType<sub>in</sub>* is the type of device to which the message will be passed. ZIL\_DEVICE\_TYPE values used by the library include: E\_CURSOR, E\_KEY, and E\_MOUSE.

## Example

```
#include <ui_evt.hpp>

ExampleFunction()
{
    // Attach the keyboard to the event manager.
    UI_DISPLAY *display = new UI_GRAPHICS_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;

    // Turn all the devices off.
    UI_EVENT event;
    event.type = D_OFF;
    eventManager->Event(event, E_DEVICE);

}

```

## UI\_EVENT\_MANAGER::First

### Syntax

```
#include <ui_evt.hpp>

UI_DEVICE *First(void);
```

### Portability

This function is available on the following environments:

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

## Remarks

This function returns a pointer to the first device in the Event Manager's list. When devices are added to the Event Manager, they are ordered so that devices with a higher priority (i.e., lower device type value) will be at the beginning of the Event Manager's list. Thus, this function will return a pointer to the highest priority device.

- *returnValue<sub>out</sub>* is the first device in the Event Manager's list.

## UI\_EVENT\_MANAGER::Get

### Syntax

```
#include <ui_evt.hpp>
```

```
virtual int Get(UI_EVENT &event, Q_FLAGS flags = Q_NORMAL);
```

### 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 an event from the Event Manager's event queue, if one is available. The event queue is used to temporarily store events that are waiting to be processed. These events can be from input devices, from the operating system (if the operating system provides an event driven messaging system), or from the programmer.

- *returnValue<sub>out</sub>* is set to 0 if an event was available and copied to the *event* argument. Otherwise, a negative value is returned, indicating that an event was not available.
- *event<sub>out</sub>* is a reference pointer to the event. This argument is a copy of the event information.
- *flags<sub>in</sub>* indicates what actions should take place when attempting to get an event from the event queue. The following flags (declared in **UI\_EVT\_HPP**) specify the available actions:

**Q\_BEGIN**—Retrieves the event from the beginning of the input queue. Setting this flag forces the Event Manager to return the oldest event in the event queue.

**Q\_BLOCK**—Remains in the `UI_EVENT_MANAGER::Get()` function polling the devices until there is an event on the queue.

**Q\_DESTROY**—Destroys the event information from the Event Manager after it is copied to *event*. **NOTE:** The `Q_NO_DESTROY` flag takes precedence over this flag.

**Q\_END**—Retrieves the event from the end of the input queue. Setting this flag forces the Event Manager to return the most recent event in the event queue.

**Q\_NO\_BLOCK**—Polls the devices and then immediately returns from the `UI_EVENT_MANAGER::Get( )` function, even if there is not an event in the event queue.

**Q\_NO\_DESTROY**—Does not destroy the event information from the input queue. If this flag is set, the next call to `UI_EVENT_MANAGER::Get()` will return the same event.

**Q\_NO\_POLL**—Does not poll the devices before checking the event queue. This is an advanced flag that should only be used by `UI_DEVICE` class objects when they communicate with the Event Manager. It prevents `UI_DEVICE` class objects from being recursively called by the `UI_EVENT_MANAGER::Get()` function.

**Q\_NORMAL**—This flag is equivalent to setting the `Q_BLOCK`, `Q_BEGIN`, `Q_DESTROY` and `Q_POLL` flags. This flag is the default if no other flag is set.

**Q\_POLL**—Ensures that all devices in the Event Manager's device list are called before information is retrieved from the event queue. This enables the devices to place any events they may have stored on the queue.

## Example

```
#include <ui_win.hpp>

main()
{

    EVENT_TYPE ccode;
    do
```

```

{
    // Get an event from the event manager.
    UI_EVENT event;
    eventManager->Get(event, Q_NORMAL);

    // Pass the event to the window manager,
    ccode = windowManager->Event(event);
} while (ccode != L_EXIT);

```

## UI EVENT MANAGER::Last

### Syntax

```

#include <ui_evt.hpp>

UI_DEVICE *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 device in the Event Manager's list. When devices are added to the Event Manager, they are ordered so that devices with a higher priority (i.e., lower device type value) will be at the beginning of the Event Manager's list. Thus, this function will return a pointer to the lowest priority device.

- *returnValue<sub>out</sub>* is the last device in the Event Manager's list.

## UI\_EVENT\_MANAGER::Put

### Syntax

```
#include <ui_evt.hpp>
```

```
virtual void Put(const UI_EVENT &event, Q_FLAGS flags = Q_END);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function puts an event into the event queue.

- *event<sub>in</sub>* is a reference pointer to the event. This argument has the event information that is put in the input queue.
- *flags<sub>in</sub>* indicates the order in which to insert the event into the event queue. The following flags (declared in **UI\_EVT.HPP**) are recognized by the **UI\_EVENT\_MANAGER::Put()** function:

**Q\_BEGIN**—Puts the event information at the beginning of the input queue (i.e., before the oldest event in the input queue.)

**Q\_END**—Puts the event information at the end of the input queue (i.e., after the most recent event in the input queue.) This flag is the default if no other flag is set.

### Example

```
#include <ui_win.hpp>

static void Exit(UI_WINDOW_OBJECT *item, UI_EVENT &event, EVENT_TYPE ccode)
{
    // Send an L_EXIT message through the system,
    event.type = L_EXIT;
```



```
    UI_EVENT_MANAGER *eventManager = item->eventManager;
    eventManager->Put(event, Q_BEGIN);
}
```

## **UI\_EVENT\_MANAGER::QFlags**

### **Syntax**

```
#include <ui_evt.hpp>

virtual Q_FLAGS QFlags(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 value of the *qFlags* member variable.

- *returnValue<sub>out</sub>* is the current flag setting for the event queue flags. If a call to **Get()** is in progress, *returnValue* will indicate the flag setting passed in the call to **Get()**. If no call to **Get()** is in process, this value will be 0.

## **UI\_EVENT\_MANAGER:-Subtract**

### **Syntax**

```
#include <ui_evt.hpp>

void Subtract(UI_DEVICE *device);
```

## Portability

This function is available on the following environments:

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

## Remarks

This function subtracts a device from the Event Manager.

- *device<sub>in</sub>* is the device to be subtracted from the Event Manager.

## UI\_EVENT\_MANAGER: operator +

### Syntax

```
#include <ui_evt.hpp>
```

```
UI_EVENT_MANAGER &operator + (UI_DEVICE *device);
```

## Portability

This function is available on the following environments:

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

## Remarks

This operator overload adds a device to the Event Manager. When devices are added to the Event Manager, they are ordered so that devices with a higher priority (i.e., lower device type value) will be at the beginning of the Event Manager's list.

- *returnValue<sub>out</sub>* is a pointer to the Event Manager. This pointer is returned so that the operator may be used in a statement containing other operations.
- *device<sub>in</sub>* is the device to be added to the Event Manager.

## **UI\_EVENT\_MANAGER::operator -**

### **Syntax**

```
#include <ui_evt.hpp>
```

```
UI_EVENT_MANAGER &operator - (UI_DEVICE *device);
```

### **Portability**

This function is available on the following environments:

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

### **Remarks**

This operator overload subtracts a device from the Event Manager.

- *returnValue<sub>out</sub>* is a pointer to the Event Manager. This pointer is returned so that the operator may be used in a statement containing other operations.
- *device<sub>in</sub>* is the device to be subtracted from the Event Manager.

## CHAPTER 13 - UI\_EVENT\_MAP

The UI\_EVENT\_MAP structure is used to map raw input device events to logical events. For example, OpenZinc Application Framework declares default event mapping for the UID\_KEYBOARD and UID\_MOUSE class objects. Some of their mapped values (in DOS) are:

<F1> — Mapped to L\_HELP; a message that causes the system to generate context-sensitive help information about the current window object.

<TAB> — Mapped to L\_NEXT; a message that moves focus to the next object on the window.

<Left-mouse-button click> — Mapped to L\_BEGIN\_SELECT if on a selection object or L\_BEGIN\_MARK if on an editable object. These messages select a window field or start a marking operation, respectively.

The UI\_EVENT\_MAP class is declared in UI\_WIN.HPP. Its public and protected members are:

```
struct ZIL_EXPORT_CLASS UI_EVENT_MAP
{
    ZIL_OBJECTID objectID;
    ZIL_LOGICAL_EVENT logicalValue;
    EVENT_TYPE eventType;
    ZIL_RAW_CODE rawCode;
    ZIL_RAW_CODE modifiers;

    static LOGICAL_EVENT MapEvent(UI_EVENT_MAP *mapTable,
        const UI_EVENT &event, ZIL_OBJECTID id1 = ID_WINDOW_OBJECT,
        ZIL_OBJECTID id2 = ID_WINDOW_OBJECT,
        ZIL_OBJECTID id3 = ID_WINDOW_OBJECT,
        ZIL_OBJECTID id4 = ID_WINDOW_OBJECT,
        ZIL_OBJECTID id5 = ID_WINDOW_OBJECT);
};
```

### General Members

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

- *objectID* is the object identification for which the match applies. (A full list of object identifications is given in **UI\_EVT.HPP**.) Each window identification has an "ID\_" prefix. Some example window object identifications are:

**ID\_WINDOW\_OBJECT**—This identification is a default identification associated with all class objects derived from the UI\_WINDOW\_OBJECT base class.

**ID\_BORDER**—This identification is associated with the UIW\_BORDER class object.

**ID\_STRING**—This identification is associated with the UIW\_STRING object or with any class object derived from the UIW\_STRING base class (e.g., UIW\_DATE, UIW\_TIME).

- *logicalValue* is the logical event to map. (A full list of logical values is given in **UI\_EVT.HPP**.) Each logical value has an "L\_" prefix. Some example logical values are:

**L\_EXIT**—Exits the application program.

**L\_BEGIN\_MARK**—Begins a mark region.

- *eventType* is the raw device identification. The following event types (declared in **UI\_EVT.HPP**) are pre-defined by OpenZinc Application Framework:

**E\_CURSOR**—Identification for the UID\_CURSOR object.

**E\_KEY**—Identification for the UID\_KEYBOARD object. This device generates keyboard input information.

**E\_MOUSE**—Identification for the UID\_MOUSE object. This device generates mouse input information.

- *rawCode* is the raw scan code or button state (depending on the type of device) of the event.
- *modifiers* is a bit field that indicates which modifier keys (i.e., shift keys, meta keys, etc.) were pressed at the time the event occurred.

## UI\_EVENT\_MAP::MapEvent

### Syntax

```
#include <ui_evt.hpp>

static LOGICAL_EVENT MapEvent(UI_EVENT_MAP *mapTable,
    const UI_EVENT &event,
    ZIL_OBJECTID id1 = ID_WINDOW_OBJECT,
    ZIL_OBJECTID id2 = ID_WINDOW_OBJECT,
    ZIL_OBJECTID id3 = ID_WINDOW_OBJECT,
    ZIL_OBJECTID id4 = ID_WINDOW_OBJECT,
    ZIL_OBJECTID id5 = ID_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 advanced function provides the logical mapping (if any) of a raw event.

- *returnValue<sub>out</sub>* is the logical event that matches the event and identification parameters. If no match occurs, this value is *event.type* (i.e., the event type passed into the **MapEvent()** function).
- *mapTable<sub>in</sub>* is a pointer to the event map table to be used by the event mapping function.
- *event<sub>in</sub>* is the raw event to be mapped. The *event.type* and *event.rawCode* values are used by the event mapping function.
- *id1<sub>in</sub>* *id2<sub>in</sub>* *id3<sub>in</sub>* *id4<sub>in</sub>* and *id5<sub>in</sub>* are hierarchal identification values used when interpreting the raw event. For example, the UIW\_TEXT class object uses the following identification values when it looks for a logical mapping:

```
id1—ID_TEXT
id2—ID_WINDOW
```

id3—ID\_WINDOW\_OBJECT  
id4—unused  
id5—unused

## Example

```
#include <ui_evt.hpp>

class ZIL_EXPORT_CLASS UI_WINDOW_OBJECT : public UI_ELEMENT
protected:
    EVENT_TYPE LogicalEvent(const UI_EVENT &event, ZIL_OBJECTID currentID)
    { return (UI_EVENT_MAP::MapEvent(eventMapTable, event, currentID,
        windowID[0], windowID[1], windowID[2], windowID[3], windowID[4])); }

EVENT_TYPE UIW_TITLE::Event(const UI_EVENT &event)
{
    EVENT_TYPE ccode = UI_WINDOW_OBJECT::LogicalEvent(event, ID_TITLE);
    switch (ccode)
    {

    }
    return (ccode);
}
```

## CHAPTER 14 - UI\_GEOMETRY\_MANAGER

The UI\_GEOMETRY\_MANAGER class object is the controlling class for geometry management. Geometry management is used to restrict an object to minimum and maximum sizes or to tie an edge of an object to other objects or its parent so that it remains at a specified distance from the object. The objects can be stretched or shrunk or can simply be repositioned. Different types of constraints can be applied to each object, allowing flexible run-time positioning and sizing.

A geometry manager should be added to the window that contains the objects being managed. If a child window has objects that should be managed, it should have its own geometry manager. Otherwise, one geometry manager is sufficient to manage all objects on a window. Constraints, such as UI\_ATTACHMENT, UI\_DIMENSION\_CONSTRAINT and UI\_RELATIVE\_CONSTRAINT, are added to the geometry manager.

The UI\_GEOMETRY\_MANAGER class is declared in UI\_WIN.HPP. Its public and protected members are:

```
class UI_GEOMETRY_MANAGER : public UI_WINDOW_OBJECT, public UI_LIST
{
public:
    static ZIL_ICHAR _className [ ] ;

    UI_GEOMETRY_MANAGER(void) ;
    virtual ~UI_GEOMETRY_MANAGER(void) ;
    virtual ZIL_ICHAR *ClassName(void) ;
    virtual          EVENT_TYPE          Event          (const          UI_EVENT          &event) ;
    virtual void information(ZIL_ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT) ;

#ifdef 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)) ;
    UI_GEOMETRY_MANAGER(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
#ifdef 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_CONSTRAINT *Add(UI_CONSTRAINT *object) ;
    UI_CONSTRAINT *Current(void) ;
    UI_CONSTRAINT *First(void) ;
};
```



```
UI_CONSTRAINT *Last(void);
UI_CONSTRAINT *Subtract(UI_CONSTRAINT *object);
UI_GEOMETRY_MANAGER &operator+(UI_CONSTRAINT *object);
UI_GEOMETRY_MANAGER &operator-(UI_CONSTRAINT *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 `UI_GEOMETRY_MANAGER` class, *\_className* is "UI\_GEOMETRY\_MANAGER."

## UI\_GEOMETRY\_MANAGER::UI\_GEOMETRY\_MANAGER

### Syntax

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

```
UI_GEOMETRY_MANAGER(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 `UI_GEOMETRY_MANAGER` class object.

## UI\_GEOMETRY\_MANAGER::~~UI\_GEOMETRY\_MANAGER

### Syntax

```
#include <ui_win.hpp>

virtual ~UI_GEOMETRY_MANAGER(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 UI\_GEOMETRY\_MANAGER object. All constraints attached to the geometry manager will also be destroyed.

## UI\_GEOMETRY\_MANAGER::Add UI\_GEOMETRY\_MANAGER::operator +

### Syntax

```
#include <ui_win.hpp>

UI_CONSTRAINT*Add(UI_CONSTRAINT*object);
    or
UI_GEOMETRY_MANAGER &operator + (UI_CONSTRAINT *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 constraint to the geometry manager. The order in which objects are added to a window is important because it affects the order in which the constraints are processed. When the window to which the geometry manager is attached is sized the geometry manager goes through its list of constraints, from first to last, calling each constraint's **Modify**( ) function. If more than one constraint can affect an object, then the constraints must be added to the geometry manager so that the constraint that needs to be processed first is added first. For example, if object 1 is tied to the left edge of object 2, using a UI\_ATTACHMENT constraint, and object 2 is tied to its parent, using a UI\_RELATIVE\_CONSTRAINT. object 2's constraint should be added to the geometry manager first. This is because its size or position will likely change if the window is sized, and object 1's position needs to be updated based on object 2's new position.

The first function adds a constraint to the geometry manager.

- *returnValue<sub>out</sub>* is a pointer to *object*.
- *object<sub>in</sub>* is a pointer to the constraint to be added to the geometry manager.

The second operator overload adds a constraint to the geometry manager. This operator overload is equivalent to calling the **UI\_GEOMETRY\_MANAGER::Add**( ) function except that it allows the chaining of constraint additions to the geometry manager.

- *returnValue<sub>out</sub>* is a pointer to the UI\_GEOMETRY\_MANAGER. This pointer is returned so that the operator may be used in a statement containing other operations.
- *object<sub>in</sub>* is a pointer to the constraint that is to be added to the geometry manager.

## UI\_GEOMETRY\_MANAGER::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<sub>out</sub>* is a pointer to *\_className*.

## UI GEOMETRY MANAGER::Current

### Syntax

```
#include <ui_win.hpp>

UI_CONSTRAINT*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 constraint, if one exists, in the geometry manager.

- *returnValue<sub>out</sub>* is a pointer to the current constraint. If there is no current constraint, *returnValue* is NULL.

## UI\_GEOMETRY\_MANAGER::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 geometry manager. It is declared virtual so that any derived geometry manager class can override its default operation.

- *returnValue<sub>out</sub>* 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<sub>in</sub>* contains a run-time message for the geometry manager. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event()**:

**S\_CHANGED, S\_CREATE and S\_MOVE**—These messages cause the window and all its sub-objects to update their size and position. If the geometry manager gets one of these messages from the window, it calls all its constraints' **Modify( )** functions.

All other messages return an S\_UNKNOWN.

## UI\_GEOMETRY\_MANAGER::First

### Syntax

```
#include <ui_win.hpp>

UI_CONSTRAINT *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 constraint, if one exists, in the geometry manager.

- *returnValue<sub>out</sub>* is a pointer to the first constraint. If there is no first constraint, *returnValue* is NULL.

## UI\_GEOMETRY\_MANAGER::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<sub>out</sub>* 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<sub>in</sub>* is a request to get or set information associated with the object. The following requests (defined in **UI\_WIN.HPP**) are recognized by the geometry manager:

**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 sent to **UI\_WINDOW\_OBJECT::Information()** for processing.

- *data<sub>in/out</sub>* 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<sub>in</sub>* 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.

## UI\_GEOMETRY\_MANAGER::Last

### Syntax

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

```
UI_CONSTRAINT *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 constraint, if one exists, in the geometry manager.

- *returnValue<sub>out</sub>* is a pointer to the last constraint. If there is no last constraint, *returnValue* is NULL.

## UI GEOMETRY MANAGER::Subtract

### Syntax

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

```
UI_CONSTRAINT*Subtract(UI_CONSTRAINT *object);  
or
```

```
UI_GEOMETRY_MANAGER &operator - (UI_CONSTRAINT"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 a constraint from the geometry manager. These functions do not delete the constraints, they merely remove them from the list. The programmer is responsible for destroying any objects explicitly subtracted from the geometry manager.

The first function subtracts a constraint from the geometry manager.



- *returnValue<sub>out</sub>* is a pointer to *object*.
- *object<sub>in</sub>* is a pointer to the constraint to be subtracted from the geometry manager.

The second operator overload subtracts a constraint from the geometry manager. This operator overload is equivalent to calling the **UI\_GEOMETRY\_MANAGER::Subtracts** ) function except that it allows the chaining of constraint subtractions from the geometry manager.

- *returnValue<sub>out</sub>* is a pointer to the UI\_GEOMETRY\_MANAGER. This pointer is returned so that the operator may be used in a statement containing other operations.
- *object<sub>in</sub>* is a pointer to the constraint that is to be subtracted from the geometry manager.

## Storage Members

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

## UI\_GEOMETRY\_MANAGER::UI\_GEOMETRY\_MANAGER

### Syntax

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

```
UI_GEOMETRY_MANAGER(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 `UI_GEOMETRY_MANAGER` by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a geometry manager 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<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* 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`."
- *object<sub>in</sub>* 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`."
- *objectTable<sub>in</sub>* 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 `UIWINDOWjOBJECT::objectTable` in "Chapter 43—`UI_WINDOW_OBJECT`" in this manual. 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<sub>in</sub>* 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 this manual. 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.

## UI\_GEOMETRY\_MANAGER::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 `UI_GEOMETRY_MANAGER` from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* 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`."
- *object<sub>in</sub>* 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`."
- *objectTable<sub>in</sub>* 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 this manual. 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<sub>in</sub>* 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 this manual. 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.

## UI\_GEOMETRY\_MANAGER::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<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* 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."
- *object<sub>in</sub>* 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."

- *objectTable<sub>in</sub>* 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 this manual. 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<sub>in</sub>* 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 this manual. 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.

## **UI GEOMETRY MANAGER::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<sub>out</sub>* is a pointer to the object's **New( )** function.

## UI\_GEOMETRY\_MANAGER::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<sub>in</sub>* is the name of the object to be stored.
- *file<sub>in</sub>* 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."
- *object<sub>in</sub>* 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."
- *objectTable<sub>in</sub>* 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 this manual. 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<sub>in</sub>* 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 this manual. 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-UI GRAPHICS DISPLAY

The `UI_GRAPHICS_DISPLAY` class implements a graphics display that uses the GFX graphics library package to draw to the screen. The `UI_GRAPHICS_DISPLAY` class is the only DOS graphics display that supports Unicode mode. Thus, if a Unicode application is being created for DOS graphics mode this display class must be used instead of any other display class. Since the `UI_GRAPHICS_DISPLAY` class is derived from `UI_DISPLAY`, only details specific to the `UI_GRAPHICS_DISPLAY` class are given in this chapter. For descriptions and examples regarding virtual or inherited display members, see "Chapter 7—UI\_DISPLAY."

The `UI_GRAPHICS_DISPLAY` class is declared in `UI_DSP.HPP`. Its public and protected members are:

```
class UI_GRAPHICS_DISPLAY : public UI_DISPLAY, public UI_REGION_LIST
{
public:
    struct GRAPHICSFONT
    {
        int font;
        int maxWidth, maxHeight;
    };

    typedef unsigned char GRAPHICSPATTERN[10];

    static UI_PATH *searchPath;
    static GRAPHICSFONT fontTable[ZIL_MAXFONTS];
    static GRAPHICSPATTERN patternTable[ZIL_MAXPATTERNS];
    UI_GRAPHICS_DISPLAY(int mode = 4);
    virtual ~UI_GRAPHICS_DISPLAY(void);
    virtual void Bitmap(ZIL_SCREENID screenID, int column, int line,
        int bitmapWidth, int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette = ZIL_NULLP(UI_PALETTE),
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
        ZIL_BITMAP_HANDLE *colorBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE),
        ZIL_BITMAP_HANDLE *monoBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE));
    virtual void BitmapArrayToHandle(ZIL_SCREENID screenID, int bitmapWidth,
        int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette, ZIL_BITMAP_HANDLE *colorBitmap,
        ZIL_BITMAP_HANDLE *monoBitmap);
    virtual void BitmapHandleToArray(ZIL_SCREENID screenID,
        ZIL_BITMAP_HANDLE colorBitmap, ZIL_BITMAP_HANDLE monoBitmap,
        int *bitmapWidth, int *bitmapHeight, ZIL_UINT8 **bitmapArray);
    virtual void Ellipse(ZIL_SCREENID screenID, int column, int line,
        int startAngle, int endAngle, int xRadius, int yRadius,
        const UI_PALETTE *palette, int fill = FALSE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void IconArrayToHandle(ZIL_SCREENID screenID, int iconWidth,
        int iconHeight, const ZIL_UINT8 *iconArray,
        const UI_PALETTE *palette, ZIL_ICON_HANDLE *icon);
    virtual void IconHandleToArray(ZIL_SCREENID screenID,
        ZIL_ICON_HANDLE icon, int *iconWidth, int *iconHeight,
        ZIL_UINT8 **iconArray);
    virtual void Line(ZIL_SCREENID screenID, int column1, int line1,
        int column2, int line2, const UI_PALETTE *palette, int width = 1,
        int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual ZIL_COLOR MapColor(const UI_PALETTE *palette, int isForeground);
};
```



```

virtual void Polygon(ZIL_SCREENID screenID, int numPoints,
    const int *polygonPoints, const UI_PALETTE *palette,
    int fill = FALSE, int _xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void Rectangle(ZIL_SCREENID screenID, int left, int top,
    int right, int bottom, const UI_PALETTE *palette, int width = 1,
    int fill = FALSE, int _xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void RectangleXORDiff(const UI_REGION koldRegion,
    const UI_REGION &newRegion, ZIL_SCREENID screenID = ID_SCREEN,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void RegionDefine(ZIL_SCREENID screenID, int left, int top,
    int right, int bottom);
virtual void RegionMove(const UI_REGION &oldRegion, int newColumn,
    int newLine, ZIL_SCREENID oldScreenID = ID_SCREEN,
    ZIL_SCREENID newScreenID = ID_SCREEN);
virtual void Text(ZIL_SCREENID screenID, int left, int top,
    const ZIL_ICHAR *text, const UI_PALETTE *palette, int length = -1,
    int fill = TRUE, int _xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextHeight(const ZIL_ICHAR *string,
    ZIL_SCREENID screenID = ID_SCREEN,
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextWidth(const ZIL_ICHAR *string,
    ZIL_SCREENID screenID = ID_SCREEN,
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int VirtualGet(ZIL_SCREENID screenID, int left, int top,
    int right, int bottom);
virtual int VirtualPut(ZIL_SCREENID screenID);

protected:
    int maxColors;
    int fillPattern;
    int backgroundColor;
    int foregroundColor;
    int fillAttributes;
    int outlineAttributes;
    signed char virtualCount;
    UI_REGION virtualRegion;
    char stopDevice;

    void SetFont(ZIL_LOGICAL_FONT logicalFont);
    void SetPattern(const UI_PALETTE *palette, int _xor);
};

```

## General Members

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

- *GRAPHICSFONT* is a structure that contains the following font information:

*font* contains the value of the font. *FNT\_SMALL\_FONT* (font is 0), *FNT\_DIALOG\_FONT* (font is 1) and *FNT\_SYSTEM\_FONT* (font is 2) are pre-defined by OpenZinc.

*maxHeight* is the height of the tallest character.

*maxWidth* is the width of the widest character.

- *GRAPHICSPATTERN* is an array of 10 bytes that make up the 8x8 bitmap pattern. The first two bytes indicate the number of rows and columns defined in the pattern. The remaining 8 bytes define the pattern. Each byte (8 bits) corresponds to 8 pixels in the pattern. The patterns defined by OpenZinc are: PTN\_SOLID\_FILL, PTN\_INTERLEAVE\_FILL and PTN\_BACKGROUND\_FILL.
- *searchPath* contains the path to be searched for the Unicode font file. This file contains character definitions for thousands of Unicode characters. The file, called UNICODE.FNT, must be found at run-time or else the application will not be able to display the proper characters.
- *fontTable* is an array of *GRAPHICFONT*. The default array contains space for 10 *ZINCFONT* entries. The following entries are pre-defined by OpenZinc:

FNT\_SMALL\_FONT—A font used to display an icon's text string.

FNT\_DIALOG\_FONT—A font used when text is displayed on window objects (e.g., UIW\_BUTTON, UIW\_STRING, UIW\_TEXT, etc.).

FNT\_SYSTEM\_FONT—A sans-serif style font used to display a window's title.

The remaining entries in *fontTable* are initially set to ROM\_8X8, which is a GFX fixed-width, 8x8, bitmapped font.

See the description of the *UI\_WINDOW\_OBJECT::font* member variable in "Chapter 43—UI\_WINDOW\_OBJECT" for information on specifying which font an object uses.

- *patternTable* is an array of *GRAPHICSPATTERN*. The default array contains space for 15 *GRAPHICSPATTERN* entries. The following entries are pre-defined by OpenZinc:

PTN\_SOLID\_FILL—Solid fill.

PTN\_INTERLEAVE\_FILL—Interleaving line fill.

PTN\_BACKGROUND\_FILL—Background fill style.

- *maxColors* is the maximum number of colors supported by the graphics mode that was initialized. For example, an EGA display might support sixteen colors. This member will be filled in according to information obtained from the GFX graphics

library after it has initialized. The GFX graphics library supports SVGA modes, including 256 color mode. OpenZinc will support whatever mode is initialized by the GFX graphics library.

- `_fillPattern` is an index into the `patternTable` specifying the current fill pattern.
- `_backgroundColor` is the current background drawing color.
- `_foregroundColor` is the current foreground drawing color.
- `_fillAttributes` is the type of filling that takes place (e.g., is the shape filled?, is a line drawn around the filled area?, etc.). This field is only used by the `UI_GRAPHICS_DISPLAY` when calling the GFX graphics functions.
- `_outlineAttributes` is the current line style. This field is only used by the `UI_GRAPHICS_DISPLAY` when calling the GFX graphics functions.
- `_virtualCount` is a count of the number of virtual screen operations that have taken place. For example, when the `VirtualGet()` function is called, `_virtualCount` is decremented. Additionally, when the `VirtualPut()` function is called, `_virtualCount` is incremented.
- `_virtualRegion` is the region affected by either `VirtualGet()` or `VirtualPut()`.
- `_stopDevice` is a variable used to disable updates of device images on the display. If `_stopDevice` is `TRUE`, no drawing will be done to the screen. Otherwise, drawing will be made directly to the screen display.

## UI\_GRAPHICS\_DISPLAY::UI\_GRAPHICS\_DISPLAY

### Syntax

```
#include <ui_dsp.hpp>
```

```
UI_GRAPHICS_DISPLAY(int mode = 4);
```

## 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 `UI_GRAPHICS_DISPLAY` object. When a new `UI_GRAPHICS_DISPLAY` class is constructed, OpenZinc sets the screen display to the background color and pattern specified by the inherited variable `backgroundPalette`.

- `modein` determines the display mode initialized (e.g., CGA, EGA, VGA, SVGA, etc.). There are several ways to initialize the `UI_GRAPHICS_DISPLAY` class. The first uses auto-selection. The following modes can be passed in to auto-select the display mode based on the graphics hardware capability:

0x01	medium resolution (320x200)
0x02	high resolution, monochrome (640x200)
0x03	EGA enhanced resolution (640x350)
0x04	VGA resolution (640x480)

Alternately, the resolution can be forced. To do this, `mode` must be **FORCE\_BIOS\_MODE** + `mode`, where `mode` can be one of the following:

0x04	320x200	4 colors
0x05	320x200	2 colors
0x06	640x200	2 colors
0x08	640x400	2 colors
0x09	720x348	2 colors
0x0D	320x200	16 colors
0x0E	640x200	16 colors
0x0F	640x350	2 colors
0x10	640x350	16 colors
0x11	640x480	2 colors
0x12	640x480	16 colors

For example, if 640x480 16-color resolution is desired, `mode` should be set to **FORCE\_BIOS\_MODE** + 0x12. **FORCE\_BIOS\_MODE** is a constant defined in **GFX.H**.

To initialize the `UI_GRAPHICS_DISPLAY` to a SVGA mode, *mode* should be set to one of the following:

0x100	640x400	256 colors
0x101	640x480	256 colors
0x102	800x600	16 colors
0x103	800x600	256 colors
0x104	1024x768	16 colors
0x105	1024x768	256 colors
0x106	1280x1024	16 colors
0x107	1280x1024	256 colors

### Example

```
#include <ui_win.hpp>
main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_GRAPHICS_DISPLAY;

    return (0);
}
```

## UI\_GRAPHICS\_DISPLAY::~UI\_GRAPHICS\_DISPLAY

### Syntax

```
#include <ui_dsp.hpp>

~UI_GRAPHICS_DISPLAY(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 `UI_GRAPHICS_`-

DISPLAY class. Care should be taken to only destroy a UI\_GRAPHICS\_DISPLAY class that is not attached to another associated object.

## **UI\_GRAPHICS\_DISPLAY::SetFont**

### **Syntax**

```
#include <ui_dsp.hpp>
```

```
void SetFont(ZIL_LOGICAL_FONT logicalFont);
```

### **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 font information used by the GFX graphics library. The information contained in the *logicalFont* entry of the *fontTable* array is used to set the font.

- *logicalFont<sub>in</sub>* is the font to be used. *logicalFont* is an entry into the *fontTable* array.

## **UI\_GRAPHICS\_DISPLAY::SetPattern**

### **Syntax**

```
#include <ui_dsp.hpp>
```

```
void SetPattern(const UI_PALETTE *palette, int _xor);
```

## 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 pattern information used by the GFX graphics library. The information contained in *palette* is used to set the pattern.

- *palette<sub>in</sub>* contains the pattern style, foreground color, and background color to be used when setting the pattern.
- *\_xor<sub>in</sub>* indicates if the pattern should be drawn with the xor attribute on. If *\_xor* is TRUE, the pattern will be an xor pattern. Otherwise, the pattern will not be xor.

## CHAPTER 16 - UI\_HELP\_STUB

The `UI_HELP_STUB` class is the base class for the help system. The help system is used to display help for the end-user. The `UI_HELP_STUB` class defines the functionality that must exist in the help system. It is an abstract class, so only classes derived from `UI_HELP_STUB`, such as `UI_HELP_SYSTEM`, can be created.

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

```
class ZIL_EXPORT_CLASS UI_HELP_STUB : public ZIL_INTERNATIONAL
{
public:
    virtual ~UI_HELP_STUB(void);
    virtual void DisplayHelp(UI_WINDOW_MANAGER *windowManager,
        UI_HELP_CONTEXT helpContext = NO_HELP_CONTEXT) = 0;
};
```

### General Members

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

### UI\_HELP\_STUB::~~UI\_HELP\_STUB

#### Syntax

```
#include <ui_win.hpp>

virtual ~UI_HELP_STUB(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 `UI_HELP_STUB`



object.

## **UI\_HELP\_STUB::DisplayHelp**

### **Syntax**

```
#include <ui_win.hpp>

virtual void DisplayHelp(UI_WINDOW_MANAGER *windowManager,
    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 pure virtual function displays the help window. See "Chapter 17—UI\_HELP\_SYSTEM" for details on the help system's implementation of this function.

- *windowManager<sub>in</sub>* is a pointer to the Window Manager.
- *helpContext<sub>in</sub>* is the help context to present. If this value is NO\_HELP\_CONTEXT, the help window system will use the default help context provided in the UI\_HELP\_SYSTEM constructor.

## CHAPTER 17 - UI HELP SYSTEM

The `UI_HELP_SYSTEM` class is used to provide help information to the end-user at runtime. The programmer can create both context-sensitive help information—so that the end-user can get help relating to the current task—and general help to aid the end-user with general aspects of the application.

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

```
class ZIL_EXPORT_CLASS UI_HELP_SYSTEM : public UI_HELP_STUB
{
public:
    static ZIL_ICHAR _className[];
    static int _defaultInitialized;

    UI_HELP_SYSTEM (ZIL_ICHAR *fileName,
        UI_WINDOW_MANAGER *windowManager = ZIL_NULLP(UI_WINDOW_MANAGER),
        UI_HELP_CONTEXT helpContext = NO_HELP_CONTEXT);
    virtual ~UI_HELP_SYSTEM(void);
    virtual void DisplayHelp(UI_WINDOW_MANAGER *windowManager,
        UI_HELP_CONTEXT helpContext = NO_HELP_CONTEXT);

    void SetLanguage(const ZIL_ICHAR *languageName);

protected:
    ZIL_STORAGE_READ_ONLY *storage;
    UIW_WINDOW *helpWindow;
    UIW_TITLE *titleField;
    UIW_TEXT *messageField;
    UI_HELP_CONTEXT defaultHelpContext;
    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 `UI_HELP_SYSTEM` class, `_className` is "UI\_HELP\_SYSTEM."
- `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. `defaultInitialized` is set to `TRUE` when the strings are set up in the object's constructor.

- *storage* is a pointer to the ZIL\_STORAGE\_READ\_ONLY that contains the help data. The **.DAT** file must be created by the programmer using the Visual Designer.
- *helpWindow* is a pointer to the UIW\_WINDOW used to display help messages. The UI\_HELP\_SYSTEM creates this window automatically.
- *titleField* is a pointer to the UIW\_TITLE object on the help window. The UI\_HELP\_SYSTEM creates this title automatically.
- *messageField* is a pointer to the UIW\_TEXT field used to display the help text. The UI\_HELP\_SYSTEM creates this field automatically.
- *defaultHelpContext* contains the default help context to be used if no other help context is specified. If the programmer wishes to use a particular help context as the default help context, he must specify the context to use in the constructor for the UI\_HELP\_SYSTEM.
- *myLanguage* is the ZIL\_LANGUAGE object that contains the string translations for this object.

## Generating help files

The help context information is read from a help **.DAT** file when needed. This file is created in the Visual Designer either directly or from a text file. For example, the text file **HELP.TXT** below was first created using a text editor and then converted into a help **.DAT** file.

```

HELP_GENERAL
General Help
This application demonstrates how to use the help system

HELP_SPECIFIC
Specific Help
The UI_HELP_SYSTEM can be used to present both context-sensitive help and
general help.

The help information can be created in a text file, which is then processed
using the Designer to produce a help file. Alternatively, the help
information can be created using the Help Context Editor in the Visual
Designer.
```

There are two help contexts in the example above. A help context consists of the context identifier, the context title, and the context help text. The help context identifier is set off by three dashes on the left side. The line below the help context name is the title that is displayed in the help window at run-time. All lines between the title and the next help context or file end are used as the help information presented for that context. The help

window will automatically do word-wrapping, so the programmer does not need to worry about line length. This means that when the help file is generated the carriage return at the end of each line in the text file is ignored. If a new line is required, place either a blank line or a backslash in the text file.

The Designer generates two files for use with help: a **.DAT** file and a **.HPP** file. The **.HPP** file should be included in each module of the program that calls the help system directly, since it contains declarations for the constants used to reference the help context information. The generated header file appears as follows:

```
#define HELP_GENERAL      0x0001  // General Help
#define HELP_SPECIFIC    0x0002  // Specific Help
```

The help context information in the text file can be modified and regenerated without recompiling the program if the help context names do not change. This is very useful if international versions of the application require different help files.

## UI\_HELP\_SYSTEM::UI\_HELP SYSTEM

### Syntax

```
#include <ui_win.hpp>
#include "filename.hpp"

UI_HELP_SYSTEM(ZIL_ICHAR *fileName,
               UI_WINDOW_MANAGER *windowManager =
               ZIL_NULLP(UI_WINDOW_MANAGER),
               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 new UI\_HELP\_SYSTEM class object.

- *fileName<sub>in</sub>* is a pointer to a string containing the name of the help **.DAT** file. This file is generated in the Visual Designer .
- *windowManager<sub>in</sub>* is a pointer to the Window Manager. It is used by the help system to display the help window.
- *helpContext<sub>in</sub>* is the help context to present when no specific help context is available. The programmer may specify the help context identifier for the help he wants used as the default.

## Example

```
#include <ui_win.hpp>
#define USE_HELP_CONTEXTS
#include "demo.hpp"

main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_MSC_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;

    UI_WINDOW_MANAGER *windowManager = new UI_WINDOW_MANAGER(display,
        eventManager);
    UI_WINDOW_OBJECT::errorSystem = new UI_ERROR_SYSTEM;
    UI_WINDOW_OBJECT::helpSystem = new UI_HELP_SYSTEM("demo.dat",
        windowManager, HELP_GENERAL);

    // Restore the system.
    delete UI_WINDOW_OBJECT::helpSystem;
    delete UI_WINDOW_OBJECT::errorSystem;
    delete windowManager;
    delete eventManager;
    delete display;
    return (0);
}
```

## UIJHELP SYSTEM:fill HELP SYSTEM

### Syntax

```
#include <ui_win.hpp>

virtual ~UI_HELP_SYSTEM(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 UI\_HELP\_SYSTEM object.

## UI\_HELP\_SYSTEM::DisplayHelp

### Syntax

```
#include <ui_win.hpp>
#include "filename.hpp"

virtual void DisplayHelp(UI_WINDOW_MANAGER *windowManager,
    UI_HELP_CONTEXT helpContext = NO_HELP_CONTEXT);
```

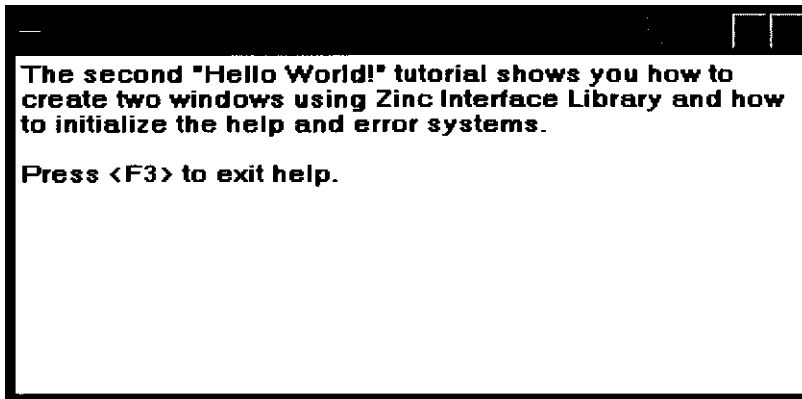
## Portability

This function is available on the following environments:

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

## Remarks

This function is used to display help information. The picture below shows a graphic representation of the UI\_HELP\_SYSTEM presentation window:



- `windowManagerin` is a pointer to the Window Manager. The help system will attach the help window to the Window Manager.
- `helpContextin` is the help context to present. If this value is `NO_HELP_CONTEXT`, the help window system will use the default help context provided in the `UI_HELP_SYSTEM` constructor.

## Example

```
#include <ui_win.hpp>
#define USE_HELP_CONTEXTS
#include "demo.hpp"

main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_GRAPHICS_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MS_MOUSE
        + new UID_CURSOR;

    UI_WINDOW_MANAGER *windowManager = new UI_WINDOW_MANAGER(display,
        eventManager);
    UI_WINDOW_OBJECT::errorSystem = new UI_ERROR_SYSTEM;
    UI_WINDOW_OBJECT::helpSystem = new UI_HELP_SYSTEM("demo.dat",
        windowManager, HELP_GENERAL);

    // Call the help system to display general help.
    windowManager->helpSystem->DisplayHelp(windowManager, HELP_SPECIFIC)
}
```

## UIW\_HELP\_SYSTEM::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 ZINC\SOURCE\INTL directory to the \ZINC\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<sub>m</sub>* is the two-letter ISO language code identifying which language the object should use.





## CHAPTER 18-UI\_ITEM

The UI\_ITEM structure is used to store different pieces of related information that, together, can be used for any of several purposes. One common use is to create several UI\_ITEM structures in an array which is then passed to the constructor of a library object. The library object constructor uses the information in each UI\_ITEM structure of the array to create an object which is added to the library object being constructed. For example, when a UI\_ITEM array is passed to a UIW\_COMBO\_BOX constructor, UIW\_STRING objects are created and attached to the combo box, thus saving the programmer from having to create the strings. OpenZinc also uses UI\_ITEM arrays as lookup tables for creating persistent objects and to maintain library strings that have been translated to different languages.

The UI\_ITEM class is declared in UI\_WIN.HPP. Its public and protected members are:

```
struct ZIL_EXPORT_CLASS UI_ITEM
{
    EVENT_TYPE value;
    void *data;
    ZIL_ICHAR *text;
    UIF_FLAGS flags;
};
```

### General Members

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

- *value* is a number associated with the item. Its use depends on the context in which the UI\_ITEM structure is used, *value* can be used to identify a particular item, or it can be an event to be put on the event queue if the item created from the UI\_ITEM structure is selected. This is common if the UI\_ITEM array is used to create UIW\_POP\_UP\_ITEMS attached to a UIW\_PULL\_DOWN\_ITEM. For example, if the following items are specified:

```
UI_ITEM _menuFlag[] =
{
    { ONE,      NULL,  " Item &one ",  MNIF_NO_FLAGS },
    { TWO,     NULL,  " Item &two ",  MNIF_NON_SELECTABLE },
    { L_EXIT,  NULL,  " E&xit",      MNIF_SEND_MESSAGE },
    { 0,      NULL,  NULL,          NULL }
};
```

and the "Exit" option is selected, an L\_EXIT event is put on the event queue.

- *data* may contain any information to associate with the item. The most common use for *data* by the library is to point to a user function associated with the object. When the object is selected, the function pointed to by *data* will be called. For example, the `_menuFlag` array defined above could be modified to contain a user function for each menu item:

```

UI_ITEM _menuFlag[] =
{
    { BTF_NO_TOGGLE,      CheckFlag,  " No toggle ",      WOF_NO_FLAGS },
    { BTF_DOWN_CLICK,    CheckFlag,  " Down click ",     WOF_NO_FLAGS },
    { BTF_CHECK_MARK,    CheckFlag,  " Checkmark ",     WOF_NO_FLAGS },
    { BTF_AUTO_SIZE,     CheckFlag,  " Auto size ",      WOF_NO_FLAGS },
    { 0,                  NULL,      NULL,                NULL }
};

EVENT_TYPE CheckFlag(UI_WINDOW_OBJECT *data, const UI_EVENT &event,
EVENT_TYPE ccode)
{
    // Typecast a pointer to the pop-up-item.
    UIW_POP_UP_ITEM *item = (UIW_POP_UP_ITEM *)data;
    switch (item->value)
    {
        case BTF_NO_TOGGLE:

            break;

        case BTF_DOWN_CLICK:

            break;
    }
    return ccode;
}

```

- *text* is text associated with the item. When creating objects using the `UI_ITEM` array, *text* is typically the text that is displayed on the screen. When used as a lookup table to obtain translated library strings, *text* will contain the text to be displayed.
- *flags* is flags associated with the item. These flags are used when constructing the class object. If the item constructed is a `UIW_POP_UP_ITEM`, *flags* is interpreted to be `MNIF_FLAGS`. If the item constructed is a `UIW_STRING`, then *flags* is interpreted to be `STF_FLAGS`. The type of object constructed from the `UI_ITEM` structure depends on what object the `UI_ITEM` array was passed to.

**NOTE:** An array of `UI_ITEM` structures should always end with an end-of-array entry indicated by a `UI_ITEM` object that has 0 or `NULL` as the *value*, *data*, *text* and *flags*. This field must be provided since no array size argument is provided:

```

UI_ITEM_buttonFlag[] =

        { 0,    NULL,    NULL,    NULL } // End of array.
};

```

## Portability

This structure is available on the following environments:

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

## Example

```

#include <ui_win.hpp>

ExampleFunction(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,    NULL }
    };

    // Create the window.
    UIW_WINDOW *window = new UIW_WINDOW(0, 0, 40, 10);
    *window
        + new UIW_BORDER
        + new UIW_TITLE(" Sample List ")
        + new UIW_PROMPT(2, 1, "List:")
        + new UIW_VT_LIST(10, 1, 20, 6, NULL, 0, listItems);
    *windowManager + window;
}

```



## CHAPTER 19-UI\_KEY

The `UI_KEY` structure is used to store the keyboard state generated by a key event. This structure maintains the keyboard's shift state and the key that was pressed.

The `UI_KEY` structure is declared in `UI_EVT.HPP`. Its public and protected members are:

```
struct ZIL_EXPORT_CLASS UI_KEY
{
    ZIL_RAW_CODE shiftState;
    ZIL_RAW_CODE value;
};
```

### General Members

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

- *shiftState* is a flag field that indicates the shift state of the keyboard. The shift state may contain one or more of the following flags (declared in `UI_EVT.HPP`):

`S_ALT`—Indicates that the <Alt> key was pressed.

`S_CAPS_LOCK`—Indicates that the <Caps-Lock> key was on.

`S_CTRL`—Indicates that the <Ctrl> key was pressed.

`S_INSERT`—Indicates that the <Ins> key was on.

`S_LEFT_SHIFT`—Indicates that the <Left-Shift> key was pressed.

`S_NUM_LOCK`—Indicates that the <Num-Lock> key was on.

`S_RIGHT_SHIFT`—Indicates that the <Right-Shift> key was pressed.

`S_SCROLL_LOCK`—Indicates that the <Scroll-Lock> key was on.

Not every shift state listed above is available on every keyboard on every environment. On some environments the keypress that generates the state is slightly different, or, as may be the case with environments that allow key mapping (e.g., Unix environments), the keypress may be completely different, depending on the

user's configuration. This list is intended to be a comprehensive list of possible states.

- *value* is the key's value. It is not the scan code.

## Portability

This structure is available on the following environments:

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

## Example

```
#include <ui_evt.hpp>

void UID_KEYBOARD::Poll()
{
    // See if a keystroke is already waiting.

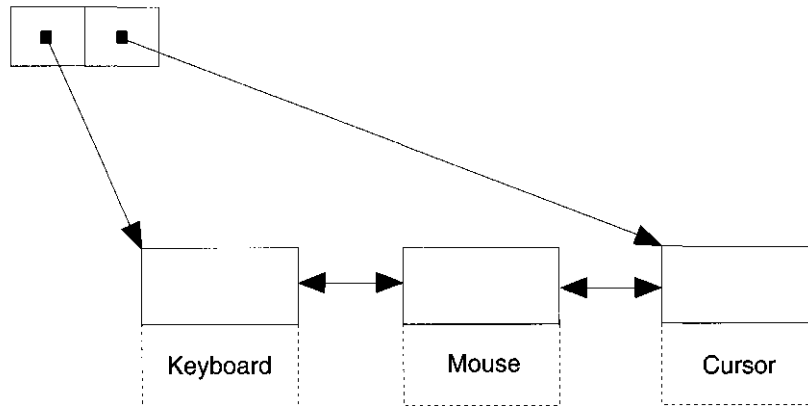
    // Get the key from the keyboard bios using INT 16H, 10H(or 00H if not
    // enhanced).
    UI_EVENT event;
    event.type = E_KEY;
    inregs.h.ah = _enhancedBios;
    ZIL_INT86(0x16, &inregs, &outregs);
    event.rawCode = outregs.x.ax;
    event.key.value = outregs.h.al;

    // Get the shift state using INT 16H, 12H(or 02H if not enhanced),
    inregs.h.ah = 0x02 + _enhancedBios;
    ZIL_INT86(0x16, &inregs, &outregs);
    event.key.shiftState = outregs.h.al;

    // Place event on the queue,
    if (state != D_OFF && eventManager)
        eventManager->Put(event, Q_END);
}
```

## CHAPTER 20 - UI LIST

The `UI_LIST` class is a container class used to manage doubly-linked list elements derived from the `UI_ELEMENT` base class. It serves as the base class to all OpenZinc Application Framework management classes (e.g., `UI_EVENT_MANAGER`, `UI_REGION_LIST`) and many control objects (e.g., `UIW_WINDOW`, `UIW_VT_LIST`). All elements in a list must be derived from the `UI_ELEMENT` class since all `UI_LIST` member functions act upon `UI_ELEMENT` class objects. The figure below illustrates how elements are linked together in a list:



The `UI_LIST` class is declared in `UI_GEN.HPP`. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UI_LIST
{
public:
    ZIL_COMPARE_FUNCTION compareFunction;

    UI_LIST(ZIL_COMPARE_FUNCTION compareFunction =
            ZIL_NULLF(ZIL_COMPARE_FUNCTION));
    virtual ~UI_LIST(void);
    UI_ELEMENT *Add(UI_ELEMENT *newElement);
    UI_ELEMENT *Add(UI_ELEMENT *element, UI_ELEMENT *newElement);
    int Count(void);
    UI_ELEMENT *Current(void);
    virtual void Destroy(void);
    UI_ELEMENT *First(void);
    UI_ELEMENT *Get(int index);
    UI_ELEMENT *Get(int (*findFunction)(void *element1, void *matchData),
                    void *matchData);
    int Index(UI_ELEMENT const *element);
    UI_ELEMENT *Last(void);
    void SetCurrent(UI_ELEMENT *element);
    virtual void Sort(void);
    UI_ELEMENT *Subtract(UI_ELEMENT *element);
    UI_LIST &operator+(UI_ELEMENT *element);
    UI_LIST &operator-(UI_ELEMENT *element);
};
```



```
protected:
    UI_ELEMENT *first, *last, *current;
};
```

## General Members

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

- *compareFunction* is a programmer defined function that will be called by the library when sorting the list of objects. *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<sub>out</sub>* 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<sub>in</sub>* 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<sub>in</sub>* is a pointer to the second element to be compared. This void pointer must be typecast according to the type of object being sorted.

- *first* and *last* point to the first and last elements in the list, respectively.
- *current* points to the current element in the list.

## UI\_LIST::UI\_LIST

### Syntax

```
#include <ui_gen.hpp>

UI_LIST(ZIL_COMPARE_FUNCTION compareFunction =
        ZIL_NULLF(ZIL_COMPARE_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 UI\_LIST object.

- *compareFunction<sub>in</sub>* is a function that is used to determine the order of list elements. See the description of the *UI\_LIST::compareFunction* member variable above for more information about the compare function.

### Example

```
#include <ui_gen.hpp>

int ButtonValueCompare(void *button1, void *button2)
{
    return(((UIW_BUTTON Mbutton1)->value - ((UIW_BUTTON *)button2)->value);
}

ExampleFunction()
{
    // Each declaration below calls the UI_LIST constructor.
    UI_LIST list1;
    UI_LIST *list2 = new UI_LIST;
    UI_LIST list3(ButtonValueCompare);
    UI_LIST *list4 = new UI_LIST(ButtonValueCompare);

    // Call the destructor for lists 2 and 4. The list1 and list3 destructors
    // are automatically called when the scope of this function ends,
    delete list2;
    delete list4;
}
```

## UI\_LIST::~~UI\_LIST

### Syntax

```
#include <ui_gen.hpp>

virtual ~UI_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 UI\_LIST object. It also destroys each element in the list.

### Example

```
#include <ui_gen.hpp>

int ButtonValueCompare(void *button1, void *button2)
{
    return(((UIW_BUTTON *)button1)->value - ((UIW_BUTTON *)button2)->value);
}

ExampleFunction()
{
    // Each declaration below calls the UI_LIST constructor.
    UI_LIST list1;
    UI_LIST *list2 = new UI_LIST;
    UI_LIST list3(ButtonValueCompare);
    UI_LIST *list4 = new UI_LIST(ButtonValueCompare);

    // Call the destructor for lists 2 and 4. The list1 and list3 destructors
    // are automatically called when the scope of this function ends,
    delete list2;
    delete list4;
}
```

## UI\_LIST::Add UI\_LIST::operator +

### Syntax

```
#include <ui_gen.hpp>

UI_ELEMENT *Add(UI_ELEMENT *newElement);
    or
UI_ELEMENT *Add(UI_ELEMENT *element, UI_ELEMENT *newElement);
    or

UI_LIST <<feoperator + (UI_ELEMENT * element);
```

### 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 element to the UI\_LIST object.

The first overloaded function adds a new element to the UI\_LIST object into a position specified by the list's *compareFunction*. If no compare function is specified when the list is constructed, *newElement* is added to the end of the list.

- *returnValue<sub>out</sub>* is a pointer to *newElement* if the addition was successful. Otherwise, *returnValue* is NULL.
- *newElement<sub>in</sub>* is a pointer to the element to be added to the list. This argument must be a class object derived from the UI\_ELEMENT base class.

The second overloaded function overrides the list's *compareFunction* by inserting *newElement* directly before *element*. The UI\_LIST::Sort( ) function may be called to sort the list when this function is used.

- *returnValue<sub>out</sub>* is a pointer to *newElement* if the addition was successful. Otherwise, *returnValue* is NULL.

- *element<sub>in</sub>* is a pointer to an element before which the new element is to be placed. If this variable is NULL, the function adds *newElement* to the end of the list.
- *newElement<sub>in</sub>* is a pointer to the element to be added to the list. This argument must be a class object derived from the UI\_ELEMENT base class.

The third operator overload adds an element to the UI\_LIST object. This operator overload is equivalent to calling the **UI\_LIST::Add()** function except that it allows the chaining of list element additions to the UI\_LIST object.

- *returnValue<sub>out</sub>* is a reference pointer to the UI\_LIST object. Returning the pointer to the UI\_LIST object allows chaining of the **UI\_LIST::operator+** overload operator.
- *element<sub>in</sub>* is a pointer to the new element that is to be added to the list.

**NOTE:** The **Add()** function and the **+ operator** are also implemented in several classes derived from UI\_LIST. Their argument types and return value types may be different than those shown here. See the appropriate chapter for more information on the derived implementation of these functions.

## 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;
}
```

## UI LIST::Count

### Syntax

```
#include <ui_gen.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 elements in the list.

- *returnValue*<sub>out</sub> is the number of elements in the list.

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    UI_LIST list1;
    list1.Add(ZIL_NULLF(ZIL_COMPARE_FUNCTION), new SAMPLE_UI_ELEMENT);
    list1.Add(ZIL_NULLF(ZIL_COMPARE_FUNCTION), new SAMPLE_UI_ELEMENT);

    // Count the number of elements in the list,
    int count = list1.Count();

}
```

## UI\_LIST::Current

### Syntax

```
#include <ui_gen.hpp>

UI_ELEMENT *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 element, if one exists, in the list.

- *returnValue<sub>out</sub>* is a pointer to the current element in the list. If there is no current element, *returnValue* is NULL.

**NOTE:** The **Current()** function is also implemented in several classes derived from **UI\_LIST**, such as **UIW\_COMBO\_BOX** and **UIW\_WINDOW**. Their argument types and return value types may be different than those shown here. See the appropriate chapter for more information on the derived implementation of these functions.

## UI\_LIST::Destroy

### Syntax

```
#include <ui_gen.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 each element in the **UI\_LIST** object, then clears the *first*, *last* and *current* members. The list's *compareFunction* remains unchanged.

## Example 1

```
#include <ui_gen.hpp>

ExampleFunction1(UI_ELEMENT *element1)
{
    UI_LIST list1;
    list1.Add(element1);

    // Destroy all the elements of the list,
    list1.Destroy();
}
}
```

## Example 2

```
ExampleFunction2(UI_ELEMENT *element1, UI_ELEMENT *element2)
{
    UI_LIST *list2 = new LIST;
    *list2 + element1 + element2;

    // Destructively remove all items from the list. The
    // element destructor is called for each item in the list.
    // Notice we have to also call delete on the list, since it was
    // dynamically constructed.
    list2->Destroy();
    delete list2;
}
}
```

## UI LIST::First

### Syntax

```
#include <ui_gen.hpp>

UI_ELEMENT *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 element, if one exists, in the list.

- *returnValue<sub>out</sub>* is a pointer to the first element in the list. If there is no first element, *returnValue* is NULL.

**NOTE:** The **First()** function is also implemented in several classes derived from **UI\_LIST**, such as **UIW\_COMBO\_BOX** and **UIW\_WINDOW**. Their argument types and return value types may be different than those shown here. See the appropriate chapter for more information on the derived implementation of these functions.

## UI\_LIST::Get

### Syntax

```
#include <ui_gen.hpp>
```

```
UI_ELEMENT *Get(int index);
```

*or*

```
UI_ELEMENT *Get(int (*findFunction)(void *element, void *matchData),  
void *matchData)\
```

## 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 specific list element.

The first overloaded function returns the list element specified by *index*. The first element in the list has an index value of 0. If the index value is invalid, NULL is returned.

- *returnValue<sub>out</sub>* is a pointer to the matching element of the list. This value is NULL if no element matched the index value.
- *index<sub>in</sub>* is the index of the list element to find. List element indexes are zero based (i.e., the first element in a list has an index value of 0).

The second overloaded function searches the UI\_LIST object for a pattern matched by *findFunction*.

- *returnValue<sub>out</sub>* is a pointer to the matching list element. This value is NULL if no element matches *matchData*.
- *findFunction<sub>in</sub>* is a pointer to a programmer-supplied function that compares a specified element with the typecast *matchData*. If an exact match is made this function must return a 0. Any non-zero value indicates that no match was made.
- *matchData<sub>in</sub>* is a pointer to the data to be matched. This can point to any data the programmer desires to match. The **Get()** function will call *findFunction* with this argument as the *matchData* parameter.

## Example 1

```
#include <ui_gen.hpp>

ExampleFunction1()
{
    UI_LIST list;
    list+newUI_ITEM("Item1")+newUI_ITEM("Item2");

    // Get the 2nd element in the list.
    UI_ELEMENT *element = list.Get(1);
    // Get the element that matches the "Item2" pattern.
    UI_ITEM *item = (ITEM *) list.Get (UI_ITEM -.Find, "Item2");
}
}
```

## Example 2

```
FindElement(void *element1, void *element2)
{
    return ((element1 == element2) ? 0 : -1);
}

ExampleFunction2()
{
    UI_LIST list2;
    ITEM *item;
    *list2
        + new ITEM("Item3")
        + (item = new ITEM("Item1"))
        + new ITEM("Item2");

    // Get the first element in the list.
    ITEM *item = (ITEM *)list2.Get(0);

    // See if item is still in the list,
    if (list2.Get(FindElement, item))
        cout << "Item1 was found in the list.";
    else
        cout << "Item1 was NOT found in the list.";
}
```

## UI\_LIST::Index

### Syntax

```
#include <ui_gen.hpp>

int Index(UI_ELEMENT 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 element. If no element matches the specified element, -1 is returned.

- *returnValue<sub>out</sub>* gives the index of the element in the UI\_LIST 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 UI\_LIST object, -1 is returned.
- *element<sub>in</sub>* is a pointer to the list element to find, *element* must be derived from UIELEMENT.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    UI_LIST list;
    ITEM *item3 = new ITEM("Item3");
    ITEM *item1 = new ITEM("Item1");
    ITEM *item2 = new ITEM("Item2");
    list + item3 + item1 + item2;

    list.Sort();
    // Get the index number of an element in a sorted list.
    cout << "Item1 is item #" << list.Index(item1) + 1 << "in the list.";
}

```

## UI LIST::Last

### Syntax

```
#include <ui_gen.hpp>

UI_ELEMENT *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 element, if one exists, in the list.

- *returnValue<sub>out</sub>* is a pointer to the last element in the list. If there is no last element, *returnValue* is NULL.

**NOTE:** The **Last()** function is also implemented in several classes derived from UI\_LIST, such as UIW\_COMBO\_BOX and UIW\_WINDOW. Their argument types and return value types may be different than those shown here. See the appropriate chapter for more information on the derived implementation of these functions.

## **UI\_LIST::SetCurrent**

### **Syntax**

```
#include <ui_gen.hpp>

void SetCurrent(UI_ELEMENT *element);
```

### **Portability**

This function is available on the following environments:

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

### **Remarks**

This function is used to set the current item in the list.

- *element<sub>in</sub>* is a pointer to the element in the list that will become current, *element* must be a member of the list (i.e., it must have been previously added to the list.)

### **Example**

```
EVENT_TYPE COMPORT_SETUP::ResetDefaults(UI_WINDOW_OBJECT *object,
    UI_EVENT &event, EVENT_TYPE ccode)
{
    if (ccode != L_SELECT)
        return ccode;

    for (UI_WINDOW_OBJECT *window = object; window->parent;
        window = window->parent)

        COMPORT_SETUP *parentWindow = (COMPORT_SETUP *)window;
```

```

        // Reset the combo box's default current items.
        parentWindow->portField->list.SetCurrent (parentWindow->defaultPort);

    }

    return ccode;
}

```

## UI\_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 UI\_LIST object using the *compareFunction* that was assigned in the constructor. If the list has no compare function, no sort occurs.

### Example

```

#include <ui_gen.hpp>
ExampleFunction()
{
    UI_LIST list (ITEM::Compare);
    ITEM *item3 = new ITEM ("Item3")
    ITEM *item1 = new ITEM ("Item1")
    ITEM *item2 = new ITEM ("Item2")
    list + item3 + item1 + item2;

    // Sort a list of items,
    list.Sort();
}

```

## UI\_LIST::Subtract UI\_LIST::operator -

### Syntax

```
#include <ui_gen.hpp>

UI_ELEMENT *Subtract(UI_ELEMENT element);
    or

UI_LIST &operator - (UI_ELEMENT *element);
```

### 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 UI\_LIST object.

The first function removes an element from the UI\_LIST object but does not call the destructor associated with the element. The programmer is responsible for deletion of each object explicitly subtracted from a list.

- *returnValue<sub>out</sub>* is a pointer to the next element in the list. This value is NULL if there are no more elements after the subtracted element.
- *element<sub>in</sub>* is a pointer to the element to be subtracted from the list.

The second operator overload removes an element from the UI\_LIST object but does not call the destructor associated with the element. This operator overload is equivalent to calling the **Subtract()** function, except that it allows the chaining of list element removals from the UI\_LIST object.

- *returnValue<sub>out</sub>* is a reference pointer to the UI\_LIST object. Returning the pointer to the list allows chaining of the **UI\_LIST::operator-** overload operator.
- *element<sub>in</sub>* is a pointer to the element that is to be removed from the list, *element* must be derived from UI\_ELEMENT.

**NOTE:** The **Subtract()** function and the **- operator** are also implemented in several classes derived from **UI\_LIST**. Their argument types and return value types may be different than those shown here. See the appropriate chapter for more information on the derived implementation of these functions.

### Example 1

```
#include <ui_gen.hpp>

ExampleFunction1(UI_ELEMENT *element1)
{
    // Construct a list, then add elements to it.
    UI_LIST list1;
    list1.Add(element1);

    // Delete a particular element from a list,
    list1.Subtract(element1);
    delete element1;

}
```

### Example 2

```
ExampleFunction2(UI_ELEMENT *element1, UI_ELEMENT *element2)
{
    // Construct a list, then add elements to it using the
    // + operator overload.
    UI_LIST *list1 = new UI_LIST;
    *list1 + element1 + element2;

    // Move elements from list1 to list2.
    UI_LIST *list2 = new UI_LIST;

    while (list1->First())
    {
        UI_ELEMENT *element = list1->First();
        *list1 - element;
        *list2 + element;
    }

}
```



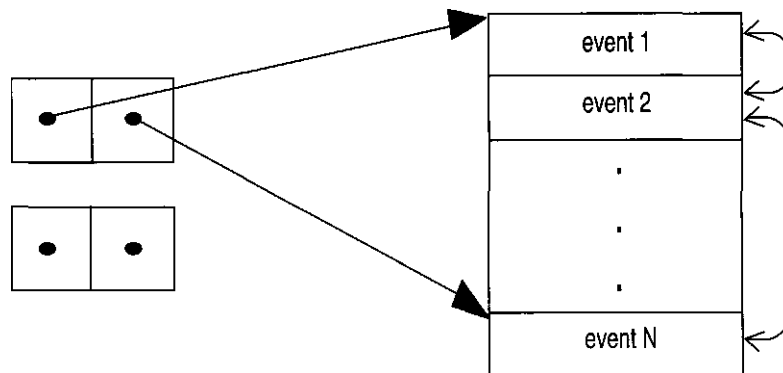


## CHAPTER 21 - UI\_LIST\_BLOCK

The `UI_LIST_BLOCK` class is used when a doubly-linked list is required but speed is also a concern. The `UI_LIST_BLOCK` is created as an array of list elements. When adding elements to and subtracting elements from the list, memory is not allocated, but instead pointers to existing elements are manipulated.

Since OpenZinc Application Framework uses lists and list elements exclusively, a `UI_LIST_BLOCK` array is structured to behave like a list so that it can access OpenZinc Application Framework objects and functions.

The `UI_LIST_BLOCK` class uses two pointers, one for the entire list and another for the list of elements that are not in use, called the free list. When a list block is initialized, an array of items is created, with the free list comprising the entire list, since it is initially empty. Each of the elements in the array is derived from the `UI_ELEMENT` base class; therefore, each has a *previous* and a *next* pointer. The figure below illustrates this arrangement:



For example, the Event Manager uses an array of event elements to store event information. This array is essentially a block of `UI_EVENT` structures. In the case of `UI_EVENT`, however, which is not derived from `UI_ELEMENT`, a `UI_QUEUE_ELEMENT` class is constructed that is derived from `UI_ELEMENT`:

```
class ZIL_EXPORT_CLASS UI_QUEUE_ELEMENT : public UI_ELEMENT
{
public:
    UI_QUEUE_ELEMENT(void);
    ~UI_QUEUE_ELEMENT(void);
    UI_EVENT event;
```

```

        UI_QUEUE_ELEMENT *Next(void);
        UI_QUEUE_ELEMENT *Previous(void);
};

```

This class is actually only an event with a *previous* and a *next* pointer, which allows an array to be set up that behaves like a list.

Every time a new element is added to a list, instead of having to allocate memory for it—an expensive operation—the space is taken from the array. Originally the entire list is in the free list, but if the array/list needs a new element, an element will be shifted out of the free list. When elements are removed from the list, instead of memory being deallocated—also an expensive operation—the element will be placed back in the free list.

Constructing arrays that act as lists allows for greater speed and efficiency within OpenZinc Application Framework. For example, the Event Manager is continually feeding information into the system. If space had to be allocated for each of these events, the application would be extremely inefficient. Instead, a large block of memory is allocated when the program is initializing, and no more allocations are made for that list block. The downside, of course, is that some memory that may not be used is permanently allocated for the list.

The `UI_LIST_BLOCK` class is declared in `UI_GEN.HPP`. Its public and protected members are:

```

class ZIL_EXPORT_CLASS UI_LIST_BLOCK : public UI_LIST
{
public:
    virtual ~UI_LIST_BLOCK(void);
    UI_ELEMENT *Add(void);
    UI_ELEMENT *Add(UI_ELEMENT *element);
    int Full(void);
    UI_ELEMENT *Subtract(UI_ELEMENT *element);

protected:
    int noOfElements;
    void *elementArray;
    UI_LIST freeList;

    UI_LIST_BLOCK(int noOfElements,
                  ZIL_COMPARE_FUNCTION compareFunction =
                    ZIL_NULLF(ZIL,,COMPARE_FUNCTION));
};

```

## General Members

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

- *noOfElements* indicates how many elements should be allocated for the list.

- *elementArray* is a pointer to the allocated block. This array must be allocated by the programmer through a derived class of UI\_LIST\_BLOCK.
- *freeList* contains pointers to the elements not currently in use.

## UI\_LIST\_BLOCK::UI\_LIST\_BLOCK

### Syntax

```
#include <ui_gen.hpp>
```

```
UI_LIST_BLOCK(int noOfElements, ZIL_COMPARE_FUNCTION compareFunction =  
ZIL_NULLF(ZIL_COMPARE_FUNCTION));
```

### 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 UI\_LIST\_BLOCK object. The constructor is protected because only derived classes should use the UI\_LIST\_BLOCK functionality.

- *noOfElements<sub>n</sub>* is the number of elements in the list.
- *compareFunction<sub>n</sub>* is a function used by the list to determine the order of each element in the list. *compareFunction* is called each time a new element is added and when the list is sorted explicitly by calling its **Sort()** member function. *compareFunction* can be provided by the programmer to allow sorting based on a key unique to the elements being placed in the list.

The definition of the *compareFunction* is as follows:

```
int FunctionName(void *element1, void *element2);
```

*returnValue<sub>out</sub>* 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<sub>in</sub>* is a pointer to the first element to be compared. This void pointer must be typecast according to the type of derived object being sorted.

*element2<sub>in</sub>* is a pointer to the second element to be compared. This void pointer must be typecast according to the type of derived object being sorted.

## Example

```
#include <ui_evt.hpp>

UI_QUEUE_BLOCK::UI_QUEUE_BLOCK(int _noOfElements) :
    UI_LIST_BLOCK(_noOfElements)
{
    // Initialize the queue block.
    UI_QUEUE_ELEMENT *queueBlock = new UI_QUEUE_ELEMENT[_noOfElements];
    elementArray = queueBlock;
    for (int i = 0; i < _noOfElements; i++)
        freeList.Add(NULL, &queueBlock[i]);
}
```

## UI\_LIST\_BLOCK::~~UI\_LIST\_BLOCK

### Syntax

```
#include <ui_gen.hpp>

virtual ~UI_LIST_BLOCK(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 UI\_LIST\_BLOCK object. It also destroys each element in the list.

## Example

```
#include <ui_evt.hpp>

UI_QUEUE_BLOCK::~UI_QUEUE_BLOCK()
{
    // Free the queue block.
    UI_QUEUE_ELEMENT *queueBlock = (UI_QUEUE_ELEMENT *)elementArray;
    delete [noOfElements]queueBlock;
}
```

## UI\_LIST\_BLOCK::Add

### Syntax

```
#include <ui_gen.hpp>

UI_ELEMENT *Add(void);
    or
UI_ELEMENT *Add(UI_ELEMENT * element);
```

### 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 element to the UI\_LIST\_BLOCK object.

The first overloaded function adds a new, empty element to the used list. The element's position is specified by the list's *compareFunction*. If no compare function is specified when the list is constructed, the element is added to the end of the list. The new element is transferred from the free list.

The second overloaded function overrides the list's *compareFunction* by inserting the new element directly before *element*.

- *returnValue<sub>out</sub>* is a pointer to the new element if the addition was successful. Otherwise, *returnValue* is NULL.

- *element<sub>in</sub>* is a pointer to an element before which the new element is to be placed. If this argument is NULL, the function adds the new element to the end of the list.

## Example

```
#include "ui_evt.hpp"

void UI_EVENT_MANAGER::Put(const UI_EVENT &event, Q_FLAGS flags)
{
    // Place the event back in the event queue.
    UI_QUEUE_ELEMENT *element =
        (UI_QUEUE_ELEMENT *)queueBlock.Add(FlagSet(flags, Q_END) ? NULL :
        queueBlock.First());
    if (element)
        element->event = event;
}
```

## UI\_LIST\_BLOCK::Full

### Syntax

```
#include <ui_gen.hpp>
```

```
int Full(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 indicates if the UI\_LIST\_BLOCK is full.

- *returnValue<sub>out</sub>* indicates if the UI\_LIST\_BLOCK is full. If the list block is full *returnValue* is TRUE. Otherwise it is FALSE.

## UI\_LIST\_BLOCK::Subtract

### Syntax

```
#include <ui_gen.hpp>
```

```
UI_ELEMENT *Subtract(UI_ELEMENT *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 removes an element from the used list and puts it back into the free list. This function does not call the destructor associated with the element.

- *returnValue<sub>out</sub>* is a pointer to the next element in the list. This value is NULL if there are no more elements after the removed element.
- *element<sub>in</sub>* is a pointer to the element to be removed from the list.

### Example

```
#include "ui_evt.hpp"

int UI_EVENT_MANAGER::Get(UI_EVENT &event, Q_FLAGS flags)
{
    UI_DEVICE *device;
    UI_QUEUE_ELEMENT *element;
    int error = -1;

    // Stay in loop while no event conditions are met.
    do
    {
        // Call all the polled devices.
        if ( IFlagSet (flags, Q_NO_POLL) )
        {

            for (device = First(); device; device = device->Next())
                device->Poll();
        }
    } while ( !event );
}
```



```

// Get the event.
element = FlagSet (flags, Q_END) ? queueBlock.Last () :
    queueBlock.First ();
if (element)
{
    event = element->event;
    if (IFlagSet (flags, Q_NO_DESTROY) )
        queueBlock.Subtract((UI_ELEMENT *)element);
    error = 0;
}
else if (FlagSet (flags, Q_NO_BLOCK) )
    return (-2);
} while (error);

// Return the error status,
return (error);
}

```

## CHAPTER 22-UI\_MACINTOSH\_DISPLAY

The UI\_MACINTOSH\_DISPLAY class implements a graphics display that uses the Apple Macintosh graphics package to draw to the screen. Since the UI\_MACINTOSH\_DISPLAY class is derived from UI\_DISPLAY, only details specific to the UI\_MACINTOSH\_DISPLAY class are given in this chapter. For descriptions and examples regarding virtual or inherited display members, see "Chapter 7—UI\_DISPLAY."

The UI\_MACINTOSH\_DISPLAY class is declared in **UI\_DSP.HPP**. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UI_MACINTOSH_DISPLAY : public UI_DISPLAY
public:
    // Forward declaration of classes used by UI_MACINTOSH_DISPLAY.
    friend class ZIL_EXPORT_CLASS UI_WINDOW_OBJECT;

    struct MACFONT
    {
        short    font;
        Style    face;
        short    mode;
        short    size;
        FontRec  **fRec;
    };

    static MACFONT fontTable[ZIL_MAXFONTS];
    static ZIL_UINT8 patternTable[ZIL_MAXPATTERNS] [8] ;
    static RGBColor *rgbColorMap;
    static CTabHandle pixMapColorTable;

    static Boolean usedMenuID[lastMenuID + 1] ;
    static MenuHandle appleMenu;
    static UI_WINDOW_OBJECT *appleAbout;
    static UI_WINDOW_OBJECT *menuBar;

    UI_MACINTOSH_DISPLAY(void);
    virtual ~UI_MACINTOSH_DISPLAY(void);
    virtual void Bitmap(ZIL_SCREENID screenID, int column, int line,
        int bitmapWidth, int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette = ZIL_NULLP(UI_PALETTE),
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
        ZIL_BITMAP_HANDLE *colorBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE),
        ZIL_BITMAP_HANDLE *monoBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE));
    virtual void BitmapArrayToHandle(ZIL_SCREENID screenID, int bitmapWidth,
        int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette, ZIL_BITMAP_HANDLE *colorBitmap,
        ZIL_BITMAP_HANDLE *monoBitmap);
    virtual void BitmapHandleToArray(ZIL_SCREENID screenID,
        ZIL_BITMAP_HANDLE colorBitmap, ZIL_BITMAP_HANDLE monoBitmap,
        int *bitmapWidth, int *bitmapHeight, ZIL_UINT8 **bitmapArray);
    virtual void Ellipse(ZIL_SCREENID screenID, int column, int line,
        int startAngle, int endAngle, int xRadius, int yRadius,
        const UI_PALETTE *palette, int fill = FALSE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void IconArrayToHandle(ZIL_SCREENID screenID, int iconWidth,
        int iconHeight, const ZIL_UINT8 *iconArray,
        const UI_PALETTE *palette, ZIL_ICON_HANDLE *icon);
    virtual void IconHandleToArray(ZIL_SCREENID screenID,
        ZIL_ICON_HANDLE icon, int *iconWidth, int *iconHeight,
```

```

        ZIL_UINT8 **iconArray);
virtual void Line(ZIL_SCREENID screenID, int column1, int line1,
                int column2, int line2, const UI_PALETTE *palette, int width = 1,
                int xor = FALSE,
                const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual ZIL_COLOR MapColor(const UI_PALETTE *palette, int isForeground);
virtual RGBColor MapRGBColor(ZIL_COLOR fromColor);
virtual void Polygon(ZIL_SCREENID screenID, int numPoints,
                    const int *polygonPoints, const UI_PALETTE *palette,
                    int fill = FALSE, int xor = FALSE,
                    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void Rectangle(ZIL_SCREENID screenID, int left, int top,
                       int right, int bottom, const UI_PALETTE *palette, int width = 1,
                       int fill = FALSE, int xor = FALSE,
                       const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void RectangleXORDiff(const UI_REGION &oldRegion,
                              const UI_REGION &newRegion, ZIL_SCREENID screenID = ID_SCREEN,
                              const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void RegionMove(const UI_REGION &oldRegion, int newColumn,
                       int newLine, ZIL_SCREENID oldScreenID = ID_SCREEN,
                       ZIL_SCREENID newScreenID = ID_SCREEN);
virtual void Text(ZIL_SCREENID screenID, int left, int top,
                  const ZIL_ICHAR *text, const UI_PALETTE *palette, int length = -1,
                  int fill = TRUE, int xor = FALSE,
                  const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
                  ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextHeight(const ZIL_ICHAR *string,
                       ZIL_SCREENID screenID = ID_SCREEN,
                       ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextWidth(const ZIL_ICHAR *string,
                      ZIL_SCREENID screenID = ID_SCREEN,
                      ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int VirtualGet(ZIL_SCREENID screenID, int left, int top,
                      int right, int bottom);
virtual int VirtualPut(ZIL_SCREENID screenID);

protected:
    int maxColors;
};

```

## General Members

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

- *MACFONT* is a structure that contains the following font information:

*font* is the font family that defines the typeface.

*face* is the style (e.g., bold, underline, etc.) used to display the text.

*mode* is the mode used to display the text. For example, the text might be XOR'ed, or displayed in inverse colors.

*size* is the point size of the font.

*fRec* is a pointer to the Macintosh font resource that contains all the information about the font.

- *fontTable* is an array of **MACFONT**. The default array contains space for 10 **MACFONT** entries. The following entries are pre-defined by OpenZinc:

**FNT\_SMALL\_FONT**—A small font similar in size to a font that might be used to display an icon's text string.

**FNT\_DIALOG\_FONT**—A font that is used when text is displayed on window objects (e.g., `UIW_STRING`, `UIW_TEXT`, etc.)

**FNT\_SYSTEM\_FONT**—The Macintosh system font.

See the description of the *UI\_WINDOW\_OBJECT: font* member variable in "Chapter 43—UI\_WINDOW\_OBJECT" for information on specifying which font an object uses.

- *patternTable* is an array containing space for 15 pattern entries. The following entries are pre-defined by OpenZinc:

**PTN\_SOLID\_FILL**—A solid fill pattern.

**PTN\_INTERLEAVE\_FILL**—An interleaving line fill pattern. OpenZinc does not currently use this pattern on the Macintosh.

**PTN\_BACKGROUND\_FILL**—The background fill pattern. OpenZinc does not currently use this pattern on the Macintosh.

**PTN\_SYSTEM\_COLOR**—The system color used for highlighting text or list entries. The user can set this color in the Color Control Panel.

- *rgbColorMap* is an array of `RGBColor` values that define the colors available in OpenZinc.
- *pixMapColorTable* is the Macintosh color table used when drawing `PixMaps`.
- *usedMenuID* is an array that keeps track of which `menuID`'s have been used. The Macintosh limits an application to 256 pop-up menus. Each array entry corresponds to a `menuID`. If the array entry is `TRUE`, that `menuID` has been used. Otherwise, the `menuID` is available.
- *appleMenu* is a pointer to the Apple menu on the menu bar.

- *appleAbout* is a pointer to the About pop-up item in the Apple menu.
- *menuBar* is a pointer to the menu bar.
- *maxColors* is the maximum number of colors supported by the graphics mode in use by the system. This member will be filled in according to information obtained from the Macintosh. OpenZinc will support whatever mode is in use by the system.

## UI\_MACINTOSH\_DISPLAY::UI\_MACINTOSH\_DISPLAY

### Syntax

```
#include <ui_dsp.hpp>

UI_MACINTOSH_DISPLAY(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 UI\_MACINTOSH\_DISPLAY class object.

### Example

```
#include <ui_win.hpp>

main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_MACINTOSH_DISPLAY;

    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;

    UI_WINDOW_MANAGER *windowManager =
        new UI_WINDOW_MANAGER(display, eventManager);
```

```
// Restore the system,  
delete windowManager;  
delete eventManager;  
delete display;  
return (0);
```

## **UI\_MACINTOSH\_DISPLAY::UI\_MACINTOSH\_DISPLAY**

### **Syntax**

```
#include <ui_dsp.hpp>  
  
~UI_MACINTOSH_DISPLAY(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 UI\_MACINTOSH\_DISPLAY class. Care should be taken to only destroy a UI\_MACINTOSH\_DISPLAY class that is not attached to another associated object.

## **UI\_MACINTOSH\_DISPLAY::MapRGBColor**

### **Syntax**

```
#include <ui_dsp.hpp>  
  
static RGBColor MapRGBColor(ZIL_COLOR fromColor);
```

## Portability

This function is available on the following environments:

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

## Remarks

This function maps a logical color to an RGB color.

- *returnValue<sub>out</sub>* is the RGB color that was mapped to.
- *fromColor<sub>in</sub>* is the logical color for which the RGB color is desired.

## CHAPTER 23 - UI\_MSC\_DISPLAY

The UI\_MSC\_DISPLAY class implements a graphics display that uses the Microsoft MSC graphics library package to draw to the screen. Since the UI\_MSC\_DISPLAY class is derived from UI\_DISPLAY, only details specific to the UI\_MSC\_DISPLAY class are given in this chapter. For descriptions and examples regarding virtual or inherited display members, see "Chapter 7—UI\_DISPLAY."

The UI\_MSC\_DISPLAY class is declared in **UI\_DSP.HPP**. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UI_MSC_DISPLAY : public UI_DISPLAY,
    public UI_REGION_LIST
{
public:
    struct ZIL_EXPORT_CLASS MSCFONT
    {
        char *typeFace;
        char *options;
    };
    typedef unsigned char MSCPATTERN[8];

    static UI_PATH *searchPath;
    static MSCFONT fontTable[ZIL_MAXFONTS];
    static MSCPATTERN patternTable[ZIL_MAXPATTERNS];

    UI_MSC_DISPLAY(int mode = 0);
    virtual ~UI_MSC_DISPLAY(void);
    virtual void Bitmap(ZIL_SCREENID screenID, int column, int line,
        int bitmapWidth, int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette = ZIL_NULLP(UI_PALETTE),
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
        ZIL_BITMAP_HANDLE *colorBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE),
        ZIL_BITMAP_HANDLE *monoBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE));
    virtual void BitmapArrayToHandle(ZIL_SCREENID screenID, int bitmapWidth,
        int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette, ZIL_BITMAP_HANDLE *colorBitmap,
        ZIL_BITMAP_HANDLE *monoBitmap);
    virtual void BitmapHandleToArray(ZIL_SCREENID screenID,
        ZIL_BITMAP_HANDLE colorBitmap, ZIL_BITMAP_HANDLE monoBitmap,
        int *bitmapWidth, int *bitmapHeight, ZIL_UINT8 **bitmapArray);
    virtual void Ellipse(ZIL_SCREENID screenID, int column, int line,
        int startAngle, int endAngle, int xRadius, int yRadius,
        const UI_PALETTE *palette, int fill = FALSE, int _xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void IconArrayToHandle(ZIL_SCREENID screenID, int iconWidth,
        int iconHeight, const ZIL_UINT8 *iconArray,
        const UI_PALETTE *palette, ZIL_ICON_HANDLE *icon);
    virtual void IconHandleToArray(ZIL_SCREENID screenID,
        ZIL_ICON_HANDLE icon, int *iconWidth, int *iconHeight,
        ZIL_UINT8 **iconArray);
    virtual void Line(ZIL_SCREENID screenID, int column1, int line1,
        int column2, int line2, const UI_PALETTE *palette, int width = 1,
        int _xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual ZIL_COLOR MapColor(const UI_PALETTE *palette, int isForeground);
    virtual void Polygon(ZIL_SCREENID screenID, int numPoints,
        const int *polygonPoints, const UI_PALETTE *palette,
        int fill = FALSE, int _xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
```



```

virtual void Rectangle(ZIL_SCREENID screenID, int left, int top,
    int right, int bottom, const UI_PALETTE *palette, int width = 1,
    int fill = FALSE, int xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void RectangleXORDiff(const UI_REGION &oldRegion,
    const UI_REGION &newRegion, ZIL_SCREENID screenID = ID_SCREEN,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void RegionDefine(ZIL_SCREENID screenID, int left, int top,
    int right, int bottom);
virtual void RegionMove(const UI_REGION koldRegion, int newColumn,
    int newLine, ZIL_SCREENID oldScreenID = ID_SCREEN,
    ZIL_SCREENID newScreenID = ID_SCREEN);
virtual void Text(ZIL_SCREENID screenID, int left, int top,
    const ZIL_ICHAR *text, const UI_PALETTE *palette, int length = -1,
    int fill = TRUE, int xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextHeight(const ZIL_ICHAR *string,
    ZIL_SCREENID screenID = ID_SCREEN,
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextWidth(const ZIL_ICHAR *string,
    ZIL_SCREENID screenID = ID_SCREEN,
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int VirtualGet(ZIL_SCREENID screenID, int left, int top,
    int right, int bottom);
virtual int VirtualPut(ZIL_SCREENID screenID);

protected:
    int maxColors;
    signed char virtualCount;
    UI_REGION virtualRegion;
    char stopDevice;
    int fillPattern;
    int backgroundColor;
    int foregroundColor;

    void SetFont(ZIL_LOGICAL_FONT logicalFont);
    void SetPattern(const UI_PALETTE *palette, int xor);
};

```

## General Members

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

- *MSCFONT* is a structure that contains the following font information:

*typeFace* contains the string name of the font. OpenZinc uses Microsoft's Helvetica font, so for the three fonts defined by OpenZinc, *typeFace* is "Helv."

*options* contains the font characteristics. For more information see `_setfont()` in the *Microsoft Visual C++ Reference*.

- *MSCPATTERN* is an array of 8 bytes that make up the 8x8 bitmap pattern. Each byte (8 bits) corresponds to 8 pixels in the pattern. The patterns defined by OpenZinc are:

PTN\_SOLID\_FILL, PTN\_INTERLEAVE\_FILL and PTN\_BACKGROUND\_FILL. For more information see **setfillpattern()** in the *Microsoft Visual C++ Reference*.

- *searchPath* contains the path to be searched for the font file. The MSC graphics library needs to access font files at run-time so that it can draw characters in graphics mode. Because OpenZinc uses Microsoft's Helvetica font, the UI\_MSC\_DISPLAY class needs to find the HELVB.FON file at run-time. If the display class cannot find this file, graphics mode will not initialize.

- *fontTable* is an array of *MSCFONT*. The default array contains space for 10 *MSCFONT* entries. The following entries are pre-defined by OpenZinc:

FNT\_SMALL\_FONT—A font that is used to display an icon's text string.

FNT\_DIALOG\_FONT—A font that is used when text is displayed on window objects (e.g., UIW\_BUTTON, UIW\_STRING, UIW\_TEXT, etc.)

FNT\_SYSTEM\_FONT—A sans-serif style font that is used to display a window's title.

NOTE: To use these fonts, or if other "stroked" fonts are added to this table, the proper Microsoft font files must be in the current working directory or in the environment's path at run-time.

See the description of the *UI\_WINDOW\_OBJECT::font* member variable in "Chapter 43—UI\_WINDOW\_OBJECT" for information on specifying which font an object uses.

- *patternTable* is an array of *MSCPATTERN*. The default array contains space for 15 *MSCPATTERN* entries. The following entries are pre-defined by OpenZinc:

PTN\_SOLID\_FILL—Solid fill.

PTN\_INTERLEAVE\_FILL—Interleaving line fill.

PTN\_BACKGROUND\_FILL—Background fill style.

- *maxColors* is the maximum number of colors supported by the graphics mode that was initialized. For example, an EGA display might support sixteen colors. This member will be filled in according to information obtained from the MSC graphics library after it has initialized. The MSC graphics library supports SVGA modes,

including 256 color mode. OpenZinc will support whatever mode is initialized by the MSC graphics library.

- *\_virtualCount* is a count of the number of virtual screen operations that have taken place. For example, when the **VirtualGet()** function is called, *\_virtualCount* is decremented. Additionally, when the **VirtualPut( )** function is called, *\_virtualCount* is incremented.
- *\_virtualRegion* is the region affected by either **VirtualGet( )** or **VirtualPut( )**.
- *\_stopDevice* is a variable used to prevent recursive updates of device images on the display. If *\_stopDevice* is TRUE, no drawing will be done to the screen. Otherwise, drawing will be made directly to the screen display.
- *\_fillPatterri* is an index into the *patternTable* specifying the current fill pattern.
- *\_backgroundColor* is the current background drawing color.
- *\_fore groundColor* is the current foreground drawing color.

## UI\_MSC\_DISPLAY::UI\_MSC\_DISPLAY

### Syntax

```
#include <ui_dsp.hpp>

UI_MSC_DISPLAY(int mode = 0);
```

### 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 **UI\_MSC\_DISPLAY** object. When a new **UI\_MSC\_DISPLAY** class is constructed, the screen display is set to the background color and pattern specified by the inherited variable *backgroundPalette*.

- *mode<sub>m</sub>* specifies the graphics mode that should be initialized. If *mode* is 0, which is the default, the UI\_MSC\_DISPLAY class will initialize the highest resolution graphics mode possible using the MSC\_MAXRESMODE constant. For more information on the possible values for *mode*, see `_setvideomode()` in the *Microsoft Visual C++ Reference*.

## Example

```
#include <ui_win.hpp>

main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_MSC_DISPLAY;

    // Restore the system,
    delete windowManager;
    delete eventManager;
    delete display;
    return (0);
}
```

## UI\_MSC\_DISPLAY::~~UI\_MSC\_DISPLAY

### Syntax

```
#include <ui_dsp.hpp>

~UI_MSC_DISPLAY(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 UI\_MSC\_DISPLAY class. Care should be taken to only destroy a UI\_MSC\_DISPLAY class that is not attached to another associated object.

## UI\_MSC\_DISPLAY::SetFont

### Syntax

```
#include <ui_dsp.hpp>

void SetFont(ZIL_LOGICAL_FONT logicalFont);
```

### 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 font information used by the MSC graphics library. The information contained in the *logicalFont* entry of the *fontTable* array is used to set the font.

- *logicalFont<sub>in</sub>* is the font to be used. *logicalFont* is an entry into the *fontTable* array.

## UI\_MSC\_DISPLAY::SetPattern

### Syntax

```
#include <ui_dsp.hpp>

void SetPattern(const UI_PALETTE *palette, int _xor);
```

### 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 pattern information used by the MSC graphics library. The information contained in *palette* is used to set the pattern.

- *palette<sub>in</sub>* contains the pattern style, foreground color, and background color to be used when setting the pattern.
- *\_xor<sub>in</sub>* indicates if the pattern should be drawn with the xor attribute on. If *\_xor* is TRUE, the pattern will be an xor pattern. Otherwise, the pattern will not be xor.



## CHAPTER 24 - UI\_MSWINDOWS\_DISPLAY

The UI\_MSWINDOWS\_DISPLAY class implements a graphics display that uses the Microsoft Windows graphics package to draw to the screen. Since the UI\_MSWINDOWS\_DISPLAY class is derived from UI\_DISPLAY, only details specific to the UI\_MSWINDOWS\_DISPLAY class are given in this chapter. For descriptions and examples regarding virtual or inherited display members, see "Chapter 7—UI\_DISPLAY."

The UI\_MSWINDOWS\_DISPLAY class is declared in UI\_DSP.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UI_MSWINDOWS_DISPLAY : public UI_DISPLAY
{
public:
    static HDC hDC;
    static HFONT fontTable[ZIL_MAXFONTS];
    static WORD patternTable[ZIL_MAXPATTERNS][8J;

    UI_MSWINDOWS_DISPLAY(HANDLE hlnstance, HANDLE hPrevInstance,
        int nCmdShow);
    virtual ~UI_MSWINDOWS_DISPLAY(void);
    virtual void Bitmap(ZIL_SCREENID screenID, int column, int line,
        int bitmapWidth, int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette = ZIL_NULLP(UI_PALETTE),
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
        ZIL_BITMAP_HANDLE *ColorBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE),
        ZIL_BITMAP_HANDLE *monoBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE));
    virtual void BitmapArrayToHandle(ZIL_SCREENID screenID, int bitmapWidth,
        int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette, ZIL_BITMAP_HANDLE *ColorBitmap,
        ZIL_BITMAP_HANDLE *monoBitmap);
    virtual void BitmapHandleToArray(ZIL_SCREENID screenID,
        ZIL_BITMAP_HANDLE colorBitmap, ZIL_BITMAP_HANDLE monoBitmap,
        int *bitmapWidth, int *bitmapHeight, ZIL_UINT8 **bitmapArray);
    virtual void Ellipse(ZIL_SCREENID screenID, int column, int line,
        int startAngle, int endAngle, int xRadius, int yRadius,
        const UI_PALETTE *palette, int fill = FALSE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void IconArrayToHandle(ZIL_SCREENID screenID, int iconWidth,
        int iconHeight, const ZIL_UINT8 *iconArray,
        const UI_PALETTE *palette, ZIL_ICON_HANDLE *icon);
    virtual void IconHandleToArray(ZIL_SCREENID screenID,
        ZIL_ICON_HANDLE icon, int *iconWidth, int *iconHeight,
        ZIL_UINT8 **iconArray);
    virtual void Line(ZIL_SCREENID screenID, int column1, int line1,
        int column2, int line2, const UI_PALETTE *palette, int width = 1,
        int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual ZIL_COLOR MapColor(const UI_PALETTE *palette, int isForeground);
    virtual void Polygon(ZIL_SCREENID screenID, int numPoints,
        const int *polygonPoints, const UI_PALETTE *palette,
        int fill = FALSE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void Rectangle(ZIL_SCREENID screenID, int left, int top,
        int right, int bottom, const UI_PALETTE *palette, int width = 1,
        int fill = FALSE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void RectangleXORDiff(const UI_REGION &oldRegion,
```



```

        const UI_REGION knewRegion, ZIL_SCREENID screenID = ID_SCREEN,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void RegionMove(const UI_REGION &oldRegion, int newColumn,
        int newLine, ZIL_SCREENID oldScreenID = ID_SCREEN,
        ZIL_SCREENID newScreenID = ID_SCREEN);
    virtual void Text(ZIL_SCREENID screenID, int left, int top,
        const ZIL_ICHAR *text, const UI_PALETTE *palette, int length = -1,
        int fill = TRUE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
        ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
    virtual int TextHeight(const ZIL_ICHAR *string,
        ZIL_SCREENID screenID = ID_SCREEN,
        ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
    virtual int TextWidth(const ZIL_ICHAR *string,
        ZIL_SCREENID screenID = ID_SCREEN,
        ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
    virtual int VirtualGet(ZIL_SCREENID screenID, int left, int top,
        int right, int bottom);
    virtual int VirtualPut(ZIL_SCREENID screenID);

protected:
    int maxColors;
};

```

## General Members

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

- *hDC* is a handle to the current display context. *hDC* is created and destroyed in the **VirtualGet()** and **VirtualPut()** functions, respectively.
- *fontTable* is an array of font handles for Microsoft Windows. The fonts used by OpenZinc are obtained from the system. The following entries are pre-defined by OpenZinc:

**FNT\_SMALL\_FONT**—A small font similar in size to a font that might be used to display an icon's text string. MS-Windows is responsible for displaying the text on an icon, so this font is not typically used by OpenZinc.

**FNT\_DIALOG\_FONT**—A font that is used when text is displayed on window objects (e.g., UIW\_BUTTON, UIW\_STRING, UIW\_TEXT, etc.)

**FNT\_SYSTEM\_FONT**—A slightly larger font similar in size to a font that might be used to display a window's title. MS-Windows is responsible for displaying the title of a window, so this font is not typically used by OpenZinc.

See the description of the *UI\_WINDOW\_OBJECT::font* member variable in "Chapter 43—UI\_WINDOW\_OBJECT" for information on specifying which font an object uses.

- *patternTable* is an array containing space for 15 pattern entries. The following entries are pre-defined by OpenZinc:

PTN\_SOLID\_FILL—A solid fill pattern.

PTN\_INTERLEAVE\_FILL—An interleaving line fill pattern.

PTN\_BACKGROUND\_FILL—The background fill pattern.

- *maxColors* is the maximum number of colors supported by the graphics mode in use by MS-Windows. For example, a VGA display might support sixteen colors. This member will be filled in according to information obtained from Windows. OpenZinc will support whatever mode is in use by Windows.

## UI\_MSWINDOWS\_DISPLAY::UI\_MSWINDOWS\_DISPLAY

### Syntax

```
#include <ui_dsp.hpp>
```

```
UI_MSWINDOWS_DISPLAY(HANDLE hInstance, HANDLE hPrevInstance,  
int nCmdShow);
```

### 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 UI\_MSWINDOWS\_DISPLAY class object.

- *hInstance<sub>m</sub>* is the particular instance under which the application is running. For example, if an application is run twice, there are two instances of that application. This value is passed in automatically by **WinMain( )**.

- *hPrevInstance<sub>in</sub>* is the previous instance of the application. If a program is run for the first time, *hPrevInstance* is 0. This value is passed in automatically by **WinMain()**.
- *nCmdShow<sub>in</sub>* is a string containing the command line parameters. This value is passed in automatically by **WinMain()**.

## Example

```
#include <ui_win.hpp>

int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance, LPSTR lpszCmdLine,
int nCmdShow)
{
    // Initialize the display.
    UI_DISPLAY *display = new UI_MSWINDOWS_DISPLAY(hInstance, hPrevInstance,
nCmdShow);

    // Initialize the event manager.
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD + new UID_MOUSE + new UID_CURSOR;

    // Initialize the window manager.
    UI_WINDOW_MANAGER *windowManager =
        new UI_WINDOW_MANAGER(display, eventManager);

    // Clean up.
    delete windowManager;
    delete eventManager;
    delete display;
    return (0);
}
```

## UI\_MSWINDOWS\_DISPLAY::~UI\_MSWINDOWS\_DISPLAY

### Syntax

```
#include <ui_dsp.hpp>

~UI_MSWINDOWS_DISPLAY(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 `UI_MS-WINDOWS_DISPLAY` class. Care should be taken to only destroy a `UI_MS-WINDOWS_DISPLAY` class that is not attached to another associated object.



## CHAPTER 25 - UI\_NEXTSTEP\_DISPLAY

The UI\_NEXTSTEP\_DISPLAY class implements a drawing package for the NEXTSTEP environment. The UI\_NEXTSTEP\_DISPLAY uses Display PostScript to actually do the drawing. Since the UI\_NEXTSTEP\_DISPLAY class is derived from UI\_DISPLAY, only details specific to the UI\_NEXTSTEP\_DISPLAY class are given in this chapter. For descriptions and examples regarding virtual or inherited display members, see "Chapter 7—UI\_DISPLAY."

The UI\_NEXTSTEP\_DISPLAY class is declared in UI\_DSP.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UI_NEXTSTEP_DISPLAY : public UI_DISPLAY
{
public:
    // Forward declaration of classes used by UI_NEXTSTEP_DISPLAY.
    friend class ZIL_EXPORT_CLASS UI_WINDOW_OBJECT;

    struct NEXTFONT
    {
        id font;
    };

    static NEXTFONT fontTable[ZIL_MAXFONTS];
    static UI_WINDOW_OBJECT *menuBar;
    UI_NEXTSTEP_DISPLAY();
    virtual ~UI_NEXTSTEP_DISPLAY(void);
    virtual void Bitmap(ZIL_SCREENID screenID, int column, int line,
        int bitmapWidth, int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette = ZIL_NULLP(UI_PALETTE),
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
        ZIL_BITMAP_HANDLE *colorBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE),
        ZIL_BITMAP_HANDLE *monoBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE));
    virtual void BitmapArrayToHandle(ZIL_SCREENID screenID, int bitmapWidth,
        int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette, ZIL_BITMAP_HANDLE *colorBitmap,
        ZIL_BITMAP_HANDLE *monoBitmap);
    virtual void BitmapHandleToArray(ZIL_SCREENID screenID,
        ZIL_BITMAP_HANDLE colorBitmap, ZIL_BITMAP_HANDLE monoBitmap,
        int *bitmapWidth, int *bitmapHeight, ZIL_UINT8 **bitmapArray);
    virtual void Ellipse(ZIL_SCREENID screenID, int column, int line,
        int startAngle, int endAngle, int xRadius, int yRadius,
        const UI_PALETTE *palette, int fill = FALSE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void IconArrayToHandle(ZIL_SCREENID screenID, int iconWidth,
        int iconHeight, const ZIL_UINT8 *iconArray,
        const UI_PALETTE *palette, ZIL_ICON_HANDLE *icon);
    virtual void IconHandleToArray(ZIL_SCREENID screenID,
        ZIL_ICON_HANDLE icon, int *iconWidth, int *iconHeight,
        ZIL_UINT8 **iconArray);
    virtual void Line(ZIL_SCREENID screenID, int column1, int line1,
        int column2, int line2, const UI_PALETTE *palette, int width = 1,
        int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual ZIL_COLOR MapColor(const UI_PALETTE *palette, int isForeground);
    virtual void MapNSColor(const UI_PALETTE *palette, int foreground,
        NXColor *nextColor);
    virtual void Polygon(ZIL_SCREENID screenID, int numPoints,
        const int *polygonPoints, const UI_PALETTE *palette,
```

```

        int fill = FALSE, int _xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void Rectangle(ZIL_SCREENID screenID, int left, int top,
        int right, int bottom, const UI_PALETTE *palette, int width = 1,
        int fill = FALSE, int _xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void RectangleXORDiff(const UI_REGION &oldRegion,
        const UI_REGION &newRegion, ZIL_SCREENID screenID = ID_SCREEN,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void RegionMove(const UI_REGION &oldRegion, int newColumn,
        int newLine, ZIL_SCREENID oldScreenID = ID_SCREEN,
        ZIL_SCREENID newScreenID = ID_SCREEN);
virtual void RegionDefine(ZIL_SCREENID screenID, int left, int top,
        int right, int bottom);
virtual void Text(ZIL_SCREENID screenID, int left, int top,
        const ZIL_ICHAR *text, const UI_PALETTE *palette, int length = -1,
        int fill = TRUE, int _xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
        ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextHeight(const ZIL_ICHAR *string,
        ZIL_SCREENID screenID = ID_SCREEN,
        ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextWidth(const ZIL_ICHAR *string,
        ZIL_SCREENID screenID = ID_SCREEN,
        ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int VirtualGet(ZIL_SCREENID screenID, int left, int top,
        int right, int bottom);
virtual int VirtualPut(ZIL_SCREENID screenID);

protected:
    int maxColors;
};

```

## General Members

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

- *NEXTFONT* is a structure that contains the following font information:

*font* is a pointer to the font.

- *fontTable* is an array of fonts each of which contains the definition of a NEXTSTEP system font. The fonts used by OpenZinc are obtained from the system. The following entries are pre-defined by OpenZinc:

**FNT\_SMALL\_FONT**—A small font similar in size to a font that might be used to display an icon's text string. NEXTSTEP is responsible for displaying the text on an icon, so this font is not typically used by OpenZinc.

**FNT\_DIALOG\_FONT**—A font that is used when text is displayed on window objects (e.g., UIW\_BUTTON, UIW\_STRING, UIW\_TEXT, etc.)

**FNT\_SYSTEM\_FONT**—A slightly larger font similar in size to a font that might be used to display a window's title. NEXTSTEP is responsible for displaying the title of a window, so this font is not typically used by OpenZinc.

See the description of the *UI\_WINDOW\_OBJECT: font* member variable in "Chapter 43—UI\_WINDOW\_OBJECT" for information on specifying which font an object uses.

- *menuBar* is a pointer to the application's pull-down menu, if one exists.
- *maxColors* is the maximum number of colors supported by the graphics mode in use by the operating environment. This member will be filled in according to information obtained from the environment. OpenZinc will support whatever mode is in use by the environment.

**NOTE:** All member functions use the standard OpenZinc screen pixel coordinates with (0,0) being the top-left corner of the display even though Display PostScript places the origin at the lower-left corner of the display. This is done to remain consistent across platforms.

## **UI NEXTSTEP DISPLAY::UI NEXTSTEP DISPLAY**

### **Syntax**

```
#include <ui_dsp.hpp>

UI_NEXTSTEP_DISPLAY(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 UI\_NEXTSTEP\_DISPLAY class object.



## Example 2

```
#include <ui_win.hpp>

main()
{
    // Initialize the display.
    UI_DISPLAY *display = new UI_NEXTSTEP_DISPLAY;

    // Initialize the event manager.
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;

    // Initialize the window manager.
    UI_WINDOW_MANAGER *windowManager =
        new UI_WINDOW_MANAGER(display, eventManager);

    // Clean up.
    delete windowManager;
    delete eventManager;
    delete display;
    return (0);
}
```

## UI\_NEXTSTEP\_DISPLAY::~UI\_NEXTSTEP\_DISPLAY

### Syntax

```
#include <ui_dsp.hpp>

~UI_NEXTSTEP_DISPLAY(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 UI\_NEXTSTEP\_DISPLAY class. Care should be taken to only destroy a UI\_NEXTSTEP\_DISPLAY class that is not attached to another associated object.

## **UI NEXTSTEP DISPLAY::MapNSColor**

### **Syntax**

```
#include <ui_dsp.hpp>

virtual void MapNSColor(const UI_PALETTE *palette, int foreground,
    NXColor *nextColor);
```

### **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 obtain a NEXTSTEP color from a UI\_PALETTE entry.

- *palette<sub>in</sub>* is the palette from which the NEXTSTEP color is to be obtained.
- *foreground<sub>in</sub>* specifies if the palette's foreground color is to be obtained. If *foreground* is TRUE, the palette's foreground color is mapped. Otherwise, the palette's background color is mapped.
- *nextColor<sub>out</sub>* is the NEXTSTEP color mapped to from the palette entry.



## CHAPTER 26-UI\_OS2\_DISPLAY

The UI\_OS2\_DISPLAY class implements a Presentation Manager display that uses the OS/2 Presentation Manager API and the Graphics Programming Interface (GPI) to draw to the screen. Since the UI\_OS2\_DISPLAY class is derived from UI\_DISPLAY, only details specific to the UI\_OS2\_DISPLAY class are given in this chapter. For descriptions and examples regarding virtual or inherited display members, see "Chapter 7—UI\_DISPLAY."

The UI\_OS2\_DISPLAY class is declared in **UI\_DSP.HPP**. Its public and protected members are:

```
Class ZIL_EXPORT_CLASS UI_OS2_DISPLAY : public UI_DISPLAY
{
public:
    static HPS hps;
    static FONTMETRICS fontTable[ZIL_MAXFONTS];

    UI_OS2_DISPLAY(void);
    virtual ~UI_OS2_DISPLAY(void);
    virtual void Bitmap(ZIL_SCREENID screenID, int column, int line,
        int bitmapWidth, int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette = ZIL_NULLP(UI_PALETTE),
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
        ZIL_BITMAP_HANDLE *ColorBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE),
        ZIL_BITMAP_HANDLE *monoBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE));
    virtual void BitmapArrayToHandle(ZIL_SCREENID screenID, int bitmapWidth,
        int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette, ZIL_BITMAP_HANDLE *ColorBitmap,
        ZIL_BITMAP_HANDLE *monoBitmap);
    virtual void BitmapHandleToArray(ZIL_SCREENID screenID,
        ZIL_BITMAP_HANDLE colorBitmap, ZIL_BITMAP_HANDLE monoBitmap,
        int *bitmapWidth, int *bitmapHeight, ZIL_UINT8 **bitmapArray);
    virtual void Ellipse(ZIL_SCREENID screenID, int column, int line,
        int startAngle, int endAngle, int xRadius, int yRadius,
        const UI_PALETTE *palette, int fill = FALSE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void IconArrayToHandle(ZIL_SCREENID screenID, int iconWidth,
        int iconHeight, const ZIL_UINT8 *iconArray,
        const UI_PALETTE *palette, ZIL_ICON_HANDLE *icon);
    virtual void IconHandleToArray(ZIL_SCREENID screenID,
        ZIL_ICON_HANDLE icon, int *iconWidth, int *iconHeight,
        ZIL_UINT8 **iconArray);
    virtual void Line(ZIL_SCREENID screenID, int column1, int line1,
        int column2, int line2, const UI_PALETTE *palette, int width = 1,
        int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual ZIL_COLOR MapColor(const UI_PALETTE *palette, int isForeground);
    virtual void Polygon(ZIL_SCREENID screenID, int numPoints,
        const int *polygonPoints, const UI_PALETTE *palette,
        int fill = FALSE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void Rectangle(ZIL_SCREENID screenID, int left, int top,
        int right, int bottom, const UI_PALETTE *palette, int width = 1,
        int fill = FALSE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void RectangleXORDiff(const UI_REGION &oldRegion,
        const UI_REGION &newRegion, ZIL_SCREENID screenID = ID_SCREEN,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
};
```

```

virtual void RegionMove(const UI_REGION &oldRegion, int newColumn,
    int newLine, ZIL_SCREENID oldScreenID = ID_SCREEN,
    ZIL_SCREENID newScreenID = ID_SCREEN);
virtual void Text(ZIL_SCREENID screenID, int left, int top,
    const ZIL_ICHAR *text, const UI_PALETTE *palette, int length = -1,
    int fill = TRUE, int _xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextHeight(const ZIL_ICHAR *string,
    ZIL_SCREENID screenID = ID_SCREEN,
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextWidth(const ZIL_ICHAR *string,
    ZIL_SCREENID screenID = ID_SCREEN,
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int VirtualGet(ZIL_SCREENID screenID, int left, int top,
    int right, int bottom);
virtual int VirtualPut(ZIL_SCREENID screenID);

protected:
    int maxColors;

    void SetFont(ZIL_LOGICAL_FONT logicalFont);

```

## General Members

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

- *hps* is a handle to the current OS/2 presentation space, *hps* is created and destroyed in the **VirtualGet( )** and **VirtualPut( )** functions, respectively.
- *fontTable* is an array of OS/2 FONTMETRICS each of which contains the definition of an OS/2 system font. The fonts used by OpenZinc are obtained from the system. The following entries are pre-defined by OpenZinc:

**FNT\_SMALL\_FONT**—A small font similar in size to a font that might be used to display an icon's text string. OS/2 is responsible for displaying the text on an icon, so this font is not typically used by OpenZinc.

**FNT\_DIALOG\_FONT**—A font that is used when text is displayed on window objects (e.g., UIW\_BUTTON, UIW\_STRING, UIW.TEXT, etc.)

**FNT\_SYSTEM\_FONT**—A slightly larger font similar in size to a font that might be used to display a window's title. OS/2 is responsible for displaying the title of a window, so this font is not typically used by OpenZinc.

See the description of the *UI\_WINDOW\_OBJECT:font* member variable in "Chapter 43—UI\_WINDOW\_OBJECT" for information on specifying which font an object uses.

- *maxColors* is the maximum number of colors supported by the graphics mode in use by the Presentation Manager. For example, a VGA display might support sixteen colors. This member will be filled in according to information obtained from the GPI. OpenZinc will support whatever mode is in use by the Presentation Manager.

**NOTE:** All member functions use the standard OpenZinc screen pixel coordinates with (0,0) being the top-left corner of the display even though OS/2 places the origin at the lower-left corner of the display. This is done to remain consistent across platforms.

## **UI\_OS2\_DISPLAY::UI\_OS2\_DISPLAY**

### **Syntax**

```
#include <ui_dsp.hpp>

UI_OS2_DISPLAY(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 UI\_OS2\_DISPLAY class object.

### **Example**

```
#include <ui_win.hpp>

main()
{
    // Initialize the display.
    UI_DISPLAY *display = new UI_OS2_DISPLAY;

    // Initialize the event manager.
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;

    // Initialize the window manager.
```

```

    UI_WINDOW_MANAGER *windowManager =
        new UI_WINDOW_MANAGER(display, eventManager);

    // Clean up.
    delete windowManager;
    delete eventManager;
    delete display;
    return (0);
}

```

## UI\_OS2\_DISPLAY::~~UI\_OS2\_DISPLAY

### Syntax

```

#include <ui_dsp.hpp>

~UI_OS2_DISPLAY(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 UI\_OS2\_DISPLAY class. Care should be taken to only destroy a UI\_OS2\_DISPLAY class that is not attached to another associated object.

## UI\_OS2\_DISPLAY::SetFont

### Syntax

```

#include <ui_dsp.hpp>

void SetFont(ZIL_LOGICAL_FONT logicalFont);

```

## 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 font information used by the GPI graphics library. The information contained in the *logicalFont* entry of the *fontTable* array is used to set the font.

- *logicalFont<sub>in</sub>* is the font to be used. *logicalFont* is an entry into the *fontTable* array.





## CHAPTER 27 - UI\_PALETTE

The UI\_PALETTE structure is used by OpenZinc Application Framework to provide color information for different display types. A palette contains entries for monochrome text mode, color text mode, color graphics mode, black-and-white graphics mode and monochrome graphics mode. Because one palette contains information for each display type, a high level object does not need to know the display type when assigning a color. The appropriate palette field will be used by the low-level display function.

The UI\_PALETTE structure is declared in **UI\_DSP.HPP**. Its public and protected members are:

```
struct ZIL_EXPORT_CLASS UI_PALETTE
{
    //      Text mode
    ZIL_ICHAR fillCharacter;           // Fill character.
    ZIL_COLOR colorAttribute;         // Color attribute
    ZIL_COLOR monoAttribute;          // Mono attribute.

    //      Graphics mode
    LOGICAL_PATTERN fillPattern;      // Fill pattern.
    ZIL_COLOR colorForeground;         // EGA/VGA colors.
    ZIL_COLOR colorBackground;
    ZIL_COLOR bwForeground;           // Black & White colors (2 color).
    ZIL_COLOR bwBackground;
    ZIL_COLOR grayScaleForeground;    // Monochrome colors (3+ color).
    ZIL_COLOR grayScaleBackground;
```

### General Members

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

- *fillCharacter* is the text mode fill character. It is the character that will be used to fill in blank areas if the drawing function calls for the area to be filled. This field is used by text mode displays only.
- *colorAttribute* contains the foreground and background color definitions for the palette. The **attrib()** macro is used to combine the color values for this field. This field is used by text mode displays only.
- *monoAttribute* contains the foreground and background monochrome definitions for the palette. The **attrib()** macro is used to combine the color values for this field. This field is used by text mode displays only.

- *fillPattern* is the graphics fill pattern. It is the pattern that will be used to fill in areas if the drawing function calls for the area to be filled. This field is used by graphics mode displays only. The following patterns (declared in UI\_DSP.HPP) can be used:

PTN\_SOLID\_FILL—The pattern is a solid, single-color pattern.

PTN\_INTERLEAVE\_FILL—The pattern will interleave two colors.

PTN\_BACKGROUND\_FILL—The pattern is a special pattern used for the background of the display.

PTN\_SYSTEM\_COLOR—The colors specified in the palette are system color identifiers for the graphical operating system. If this pattern is specified OpenZinc will query the operating system to obtain the color defined for the color identifier.

PTN\_RGB\_COLOR—The colors specified in the palette are RGB color values. The RGB constants are defined in UI\_DSP.HPP.

- *colorForeground* and *colorBackground* are the color foreground and background values to use in color graphics mode.
- *bwForeground* and *bwBackground* are the black-and-white foreground and background values to use in black-and-white graphics mode, including Hercules displays.
- *grayScaleForeground* and *grayScaleBackground* are gray scale foreground and background values to use in monochrome graphics mode.

## 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 };

main( )
{
    display->backgroundPalette = &backgroundPalette;

}
```

## CHAPTER 28 - UI\_PALETTE\_MAP

The `UI_PALETTE_MAP` structure is used by OpenZinc Application Framework class objects for color map information. The structure associates an object identifier and a logical palette identifier with a palette. By constructing a table of `UI_PALETTE_MAP` objects it is possible to determine which palette to use for a specific object in a specific scenario. For example, a button that is non-selectable may draw differently than a button that is current. Each combination of button and specific scenario should be represented in the table of palette maps.

The `UI_PALETTE_MAP` structure is declared in `UI_WIN.HPP`. Its public and protected members are:

```
struct ZIL_EXPORT_CLASS UI_PALETTE_MAP
{
    ZIL_OBJECTID objectID;
    ZIL_LOGICAL_PALETTE logicalPalette;
    UI_PALETTE palette;

    static UI_PALETTE *MapPalette(UI_PALETTE_MAP *mapTable,
        ZIL_LOGICAL_PALETTE logicalPalette,
        ZIL_OBJECTID id1 ID_WINDOW_OBJECT,
        ZIL_OBJECTID id2 ID_WINDOW_OBJECT,
        ZIL_OBJECTID id3 ID_WINDOW_OBJECT,
        ZIL_OBJECTID id4 ID_WINDOW_OBJECT,
        ZIL_OBJECTID id5 ID_WINDOW_OBJECT);
};
```

### General Members

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

- *objectID* is a value that identifies the type of object for which the palette map entry applies. For example, all palette maps that pertain to the `UIW_BUTTON` class have an *objectID* of `ID_BUTTON`. A full list of object identifications is given in `UI_WIN.HPP`. Some example object identifications are:

**ID\_WINDOW\_OBJECT**—Associated with all class objects derived from the `UI_WINDOW_OBJECT` base class.

**ID\_BORDER**—Associated with the `UIW_BORDER` class object.

**ID\_STRING**—Associated with the `UIW_STRING` object or with any class object derived from the `UIW_STRING` base class (e.g., `UIW_DATE`, `UIW_TIME`).

- *logicalPalette* is a value that identifies the scenario for the object. For example, a button may be non-selectable, may be current or may not have any special look. The following logical palette identifications (defined in UI\_WIN.HPP) are recognized:

PM\_ACTIVE—Indicates that the object is active.

PM\_CURRENT—Indicates that the object is current.

PM\_HOT\_KEY—Indicates that the object has a hotkey character.

PM\_INACTIVE—Indicates that the object is not active.

PM\_NON\_SELECTABLE—Indicates that the object is non-selectable.

PM\_SELECTED—Indicates that the object is selected.

PM\_ANY—Indicates that the object has no special look.

PM\_SPECIAL—Indicates that the palette is a special palette.

The following algorithm is used to determine which palette map identification is used:

1—If the object is not selectable (i.e., the object's *woFlags* variable has the WOF\_NON\_SELECTABLE flag set), the PM\_NON\_SELECTABLE palette map is used.

2—If the object is in an inactive state (i.e., its parent window is not the current window) the PM\_INACTIVE color map is used.

3—If the object has been selected (i.e., the object's *woStatus* variable has the WOS\_SELECTED status set), the PM\_SELECTED color map is used.

4—If the object is not current but its parent window is the current window, the PM\_ACTIVE color map is used.

5—If the object is current and its parent window is the current window, the PM\_CURRENT color map is used.

6—If no match is found in the previous cases, PM\_ANY is used.

- *palette* is the palette associated with the object type and scenario.

## UI\_PALETTE\_MAP::MapPalette

### Syntax

```
#include <ui_map.hpp>

static UI_PALETTE *MapPalette(UI_PALETTE_MAP *mapTable,
    ZIL_LOGICAL_PALETTE logicalPalette,
    ZIL_OBJECTID id1 = ID_WINDOW_OBJECT,
    ZIL_OBJECTID id2 = ID_WINDOW_OBJECT,
    ZIL_OBJECTID id3 = ID_WINDOW_OBJECT,
    ZIL_OBJECTID id4 = ID_WINDOW_OBJECT,
    ZIL_OBJECTID id5 = ID_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 [advanced](#) function searches a palette map table for a match on the object type and the scenario for the object. It returns the palette mapped to, if a match is found.

**NOTE:** Not all environments use the palette map tables to draw the objects. In those graphical operating systems that already handle drawing the objects, OpenZinc typically lets the operating system do the drawing. However, if an object is an ownerdraw object (i.e., it has the `WOS_OWNERDRAW` status set, so it draws from the `DrawItem()` function), however, it will usually bypass the operating system and use the palette map table to draw.

- *returnValue<sub>out</sub>* is the palette that was mapped from the object identifier and the scenario.
- *mapTable<sub>in</sub>* is a pointer to the palette map table to be searched. OpenZinc uses several by default:

*\_normalPaletteMapTable* contains the normal mappings for all objects.

*\_errorPaletteMapTable* contains the mappings for the error window.

*\_helpPaletteMapTable* contains the mappings for the help window.

*UI\_DISPLAY::xorPalette* contains the mapping for XOR drawing.

*UI\_DISPLAY::backgroundPalette* contains the mappings for the background of the screen.

- *logicalPalette<sub>in</sub>* is a value that identifies the scenario for the object.
- *idl<sub>in</sub>* *id2<sub>in</sub>* *id3<sub>in</sub>* *id4<sub>in</sub>* and *id5<sub>in</sub>* are object identifiers. The five values are used to identify the object's inheritance hierarchy. Typically, the object's *windowID* array supplies these values. If **MapPalette** cannot find a match on *idl* it will attempt to find a match on *id2*, and so on until it either has a match or can't find a match. It is generally a good idea to provide a catch-all palette map for `ID_WINDOW_OBJECT` and `PM_ANY` that would be used if no other palette maps match.

## Example

```
#include <ui_win.hpp>
EVENT_TYPE UIW_BORDER::DrawItem(const UI_EVENT &event, EVENT_TYPE ccode)
{
    // Draw the border.
    int size = display->cellHeight;
    UI_REGION region = parent->trueRegion;
    UI_PALETTE *outline = LogicalPalette(ccode, ID_OUTLINE);
    display->Rectangle(screenID, region, outline, 1, FALSE, FALSE, &clip);
    display->Rectangle(screenID, trueRegion, outline, 1, FALSE, FALSE, &clip);
    --region;
    if (ccode == S_DISPLAY_ACTIVE && FlagSet(parent->woAdvancedFlags,
        WOAF_DIALOG_OBJECT))
        lastPalette = UI_PALETTE_MAP::MapPalette(paletteMapTable, PM_SELECTED,
            ID_BORDER);

    return (ccode);
}
```

## CHAPTER 29 - UI\_PATH

The UI\_PATH class is used to maintain a list of path elements. A path element, which is a UI\_PATH\_ELEMENT class object, contains a path to a specific directory. An object can use a UI\_PATH object to keep a list of directories that may contain data the object depends on. For example, a graphics display class may need run-time access to some graphics font definition files. The class could use a UI\_PATH object to search for the font files initially or, once they've been located, to keep track of their directory.

The UI\_PATH class is declared in **UI\_GEN.HPP**. Its public and protected members are:

```
Class ZIL_EXPORT_CLASS UI_PATH : public UI_LIST, public ZIL_INTERNATIONAL
{
public:
    UI_PATH(ZIL_ICHAR *programPath = ZIL_NULLP(ZIL_ICHAR),
            int rememberCWD = TRUE);
    ~UI_PATH(void);
    const ZIL_ICHAR *FirstPathName(void);
    const ZIL_ICHAR *NextPathName(void);

    // List members
    UI_PATH_ELEMENT *Current(void);
    UI_PATH_ELEMENT *First(void);
    UI_PATH_ELEMENT *Last(void);

public:
    static ZIL_ICHAR *_pathString;
    static ZIL_ICHAR *_zincPathString;
};
```

### General Members

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

- *\_pathString* is a string used to get the path from the appropriate operating system environment variable. By default, *\_pathString* is "PATH" thus allowing the PATH environment variable to be obtained.

*\_zincPathString* is a string used to get the OpenZinc data file path from the appropriate operating system environment variable. By default, *\_zincPathString* is "ZINC\_PATH" thus allowing the OpenZinc data file path to be obtained from the ZINC\_PATH environment variable. The use of the ZINC\_PATH environment variable is optional.



## UI\_PATH::UI\_PATH

### Syntax

```
#include <ui_gen.hpp>

UI_PATH(ZIL_ICHAR *programPath = ZIL_NULLP(ZIL_ICHAR),
        int rememberCWD = TRUE);
```

### 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 UI\_PATH class object.

- *programPath<sub>in</sub>* is a string containing the program's origination directory.
- *rememberCWD<sub>in</sub>* indicates if a path element containing the current working directory should be placed in the list of path elements. If *rememberCWD* is TRUE, the current working directory is placed in the list. Otherwise it is not.

### Example

```
#include <ui_gen.hpp>

main(int argc, char *argv[])
{
    // Initialize the path.
    UI_PATH *path = new UI_PATH(argv[0], TRUE);
    UI_DISPLAY *display = new UI_MSC_DISPLAY(0, 0, path);

    delete display;
    delete path;
}
```

## UI\_PATH::~~UI\_PATH

### Syntax

```
#include <ui_gen.hpp>

~UI_PATH(void);
```

### Portability

This function is available on the following environments:

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

### Remarks

This destructor destroys the class information associated with the UI\_PATH object. All path elements attached to the object will also be destroyed.

### Example

```
#include <ui_gen.hpp>

main(int argc, char *argv[])
{
    // Initialize the path.

    UI_PATH *path = new UI_PATH(argv[0], TRUE);

    delete path;
}
```

## UI\_PATH::Current

### Syntax

```
#include <ui_gen.hpp>

UI_PATH_ELEMENT *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 path element, if one exists, in the list.

- *returnValue<sub>out</sub>* is a pointer to the current path element in the list. If there is no current element, *returnValue* is NULL.

## UI\_PATH::First

### Syntax

```
#include <ui_gen.hpp>
```

```
UI_PATH_ELEMENT *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 path element, if one exists, in the list.

- *returnValue<sub>out</sub>* is a pointer to the first path element in the list. If there is no first element, *returnValue* is NULL.

## UI\_PATH::FirstPathName

### Syntax

```
#include <ui_gen.hpp>

const ZIL_ICHAR *FirstPathName(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 path or directory of the first path element.

- *returnValue<sub>out</sub>* is a pointer to a string containing the path or directory of the first path element. If there is no first element, it returns NULL.

### Example

```
#include <stdlib.h>
#include <stdio.h>
#include <ui_gen.hpp>

int ExampleFunction(const ZIL_ICHAR *file, unsigned int mode)
{
    UI_PATH *path = new UI_PATH;

    ZIL_ICHAR *pathName;
    pathName = path->FirstPathName();
    if (pathName == NULL)
        return (-1);

    else
    {
        while ((pathName = path->NextPathName()) != 0)
            if ((handle = open(pathName, mode)) >= 0)
                return (handle);
    }
    return (-1);
}
```

## **UI\_PATH::Last**

### **Syntax**

```
#include <ui_gen.hpp>

UI_PATH_ELEMENT *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 path element, if one exists, in the list.

- *returnValue<sub>out</sub>* is a pointer to the last path element in the list. If there is no last element, *returnValue* is NULL.

## **UI\_PATH::NextPath Name**

### **Syntax**

```
#include <ui_gen.hpp>

const ZIL_ICHAR *NextPathName(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 path or directory of the path element after the current path element.

- *returnValue<sub>out</sub>* is a pointer to a string containing the path or directory of the next path element. If there is no next element, it returns NULL.

## Example

```
#include <stdlib.h>
#include <stdio.h>
#include <ui_gen.hpp>

int ExampleFunction(const ZIL_ICHAR *file, unsigned int mode)
{
    int handle;
    UI_PATH *path = new UI_PATH;

    SetFileName (file) ;
    ZIL_ICHAR *pathName;
    pathName = path->FirstPathName()
    while ((pathName = path->NextPathName()) != 0)
        if ((handle = open(pathName, mode)) >= 0)
            return (handle);
    return (-1);
}
```



## CHAPTER 30-UI\_PATH\_ELEMENT

The UI\_PATH\_ELEMENT class is used to store a directory path. Some classes in OpenZinc need to keep track of the location of files on disk. By placing a path element in a linked list maintained by the UI\_PATH class, an object can maintain a path to any number of directory locations.

The UI\_PATH\_ELEMENT class is declared in **UI\_GEN.HPP**. Its public and protected members are:

```
Class ZIL_EXPORT_CLASS UI_PATH_ELEMENT : public UI_ELEMENT,
    public ZIL_INTERNATIONAL
{
public:
    ZIL_ICHAR *pathName;

    UI_PATH_ELEMENT(ZIL_ICHAR *pathName, int length = -1);
    ~UI_PATH_ELEMENT(void);

    // List members.
    UI_PATH_ELEMENT *Next(void);
    UI_PATH_ELEMENT *Previous(void);
};
```

### General Members

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

- *pathName* is a string containing the directory to be searched. *pathName* may contain drive specifiers.

### UI\_PATH\_ELEMENT::UI\_PATH\_ELEMENT

#### Syntax

```
#include <ui_gen.hpp>
```

```
UI_PATH_ELEMENT(ZIL_ICHAR *pathName, int length = -1);
```



## 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 `UI_PATH_ELEMENT` object. The `UI_PATH_ELEMENT` is used to store a path name and is used in conjunction with the `UI_PATH` class.

- *pathName<sub>in</sub>* contains the path or directory name.
- *length<sub>in</sub>* is the length of *pathName*. If no value is entered for *length*, the length of *pathName* is used.

## Example

```
#include <ui_gen.hpp>

UI_PATH::UI_PATH(ZIL_ICHAR *programPath, int rememberCWD)
{
    // Get the path names.
    for (int i = 0; i < 3; i++)
    {
        // Determine which path to look for.
        ZIL_ICHAR path[256];
        if (i == 0 && rememberCWD)
            getcwd(path, 256);
        else if (i == 1 && programPath)
            strcpy(path, programPath);
        else if (i == 2 && getenv("PATH"))
            strcpy(path, getenv("PATH"));
        else
            strcpy(path, "");

        // Parse the directory tree.
        for (int j = 0; path[j]; )
        {
            for (int k = j; path[k] && path[k] != ';'; )
                k++;
            Add(NULL, new UI_PATH_ELEMENT(spath[j], k - j));
            j = path[k] ? k + 1 : k;
        }
    }
}
```

## UI\_PATH\_ELEMENT::~~UI\_PATH\_ELEMENT

### Syntax

```
#include <ui_gen.hpp>

~UI_PATH_ELEMENT(void);
```

### Portability

This function is available on the following environments:

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

### Remarks

This destructor destroys the class information associated with the UI\_PATH\_ELEMENT object.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{

    path - pathElement;
    delete pathElement;
}
```

## UI\_PATH\_ELEMENT::Next

### Syntax

```
#include <ui_gen.hpp>

UI_PATH_ELEMENT *Next(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 next path element, if one exists, in the list of path elements.

## UI\_PATH\_ELEMENT::Previous

### Syntax

```
#include <ui_gen.hpp>
```

```
UI_PATH_ELEMENT *Previous(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 previous path element, if one exists, in the list of path elements.

## CHAPTER 31 - UI\_POSITION

The UI\_POSITION structure is used to store and manipulate screen positional information (e.g., mouse screen positions).

The UI\_POSITION structure is declared in **UI\_DSP.HPP**. Its public and protected members are:

```
struct ZIL_EXPORT_CLASS UI_POSITION
{
    int column, line;

    #if defined(ZIL_MSWINDOWS)
        void Assign(const POINT &point);
    #elif defined(ZIL_OS2)
        void Assign(const POINTL &point);
    #elif defined(ZIL_MACINTOSH)
        void Assign(const Point &point);
    #elif defined(ZIL_NEXTSTEP)
        void Assign(const NXPoint &point);
    #endif
    int operator==(const UI_POSITION deposition) const;
    int operator!=(const UI_POSITION deposition) const;
    int operator<(const UI_POSITION deposition) const;
    int operator>(const UI_POSITION &position) const;
    int operator>=(const UI_POSITION ^position) const;
    int operator<=(const UI_POSITION deposition) const;
    UI_POSITION &operator++(void);
    UI_POSITION &operator--(void);
    UI_POSITION &operator+=(int offset);
    UI_POSITION &operator-=(int offset);
};
```

### General Members

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

- *column* is the horizontal position value. This value may be in cells, minicells or pixels depending on the context of the UI\_REGION being used.
- *line* is the vertical position value. This value may be in cells, minicells or pixels depending on the context of the UI\_REGION being used.

### Example

```
#include <ui_evt.hpp>
EVENT_TYPE UID_CURSOR::Event(const UI_EVENT &event)
```

```

switch (event.rawCode)
{

case D_SHOW:
    if (state != D_OFF)
    {
        UI REGION region;
        region.left = position.column;
        region.top = position.line;
        region.right = region.left + display->cellWidth - 1;
        region.bottom = region.top + display->cellHeight - 1;
        if (region.Overlap(event.region))
            tState = (event.rawCode == D_HIDE) ? D_HIDE : D_ON;
    }
    break;

}
}

```

## UI\_POSITION::Assign

### Syntax

```

#include <ui_dsp.hpp>

void Assign(const POINT &point),
    or
void Assign(const POINTL &point);
    or
void Assign(const Point &point)\
    or

void Assign(const NXPoint &point);

```

### Portability

This function is available on the following environments:

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

## Remarks

This function copies the position information from the operating system-specific structure into the UI\_POSITION structure.

- *point<sub>in</sub>* is the operating system-specific position structure whose value is to be copied into the UI\_POSITION structure.

## Example

```
#include <ui_dsp.hpp>
ExampleFunction ( )
{
    position.Assign(tPosition);
}
}
```

## UI\_POSITION::operator ==

### Syntax

```
#include <ui_dsp.hpp>

int operator == (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 operator overload determines if the position maintained by the UI\_POSITION structure is the same as *position*.

- *returnValue<sub>out</sub>* is TRUE if the UI\_POSITION is the same as *position*. Otherwise, *returnValue* is FALSE.

- *position<sub>in</sub>* is the position to be compared.

## Example

```
#include <ui_dsp.hpp>
ExampleFunction ( )
{
    if (position == tPosition)
    {

    }
}
```

## UI\_POSITION: operator !=

### Syntax

```
#include <ui_dsp.hpp>

int operator != (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 operator overload determines if the UI\_POSITION structure is not equal to the UI\_POSITION structure specified by *position*.

- *returnValue<sub>out</sub>* is TRUE if the UI\_POSITION structure is not the same as *position*. Otherwise, *returnValue* is FALSE.
- *position<sub>in</sub>* is the position to be compared.

## Example

```
#include <ui_dsp.hpp>

ExampleFunction ( )
{
    if (position != tPosition)
    {

    }
}
```

## UI\_POSITION-operator <

### Syntax

```
#include <ui_dsp.hpp>

int operator < (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 operator overload determines whether the UI\_POSITION structure is less than the UI\_POSITION structure specified by *position*. The UI\_POSITION structure is less than *position* if the column value of the UI\_POSITION structure is less than the column value of *position* or if the line value of the UI\_POSITION structure is less than the line value of *position*.

- *returnValue*<sub>out</sub> is TRUE if the UI\_POSITION structure is less than *position*. Otherwise, *returnValue* is FALSE.
- *position*<sub>in</sub> is the position to be compared.



## Example

```
#include <ui_dsp.hpp>

ExampleFunction ( )
{
    if (position < tPosition)
    {

    }
}
```

## UI\_POSITION::operator >

### Syntax

```
#include <ui_dsp.hpp>

int operator > (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 operator overload determines whether the UI\_POSITION structure is greater than the UI\_POSITION structure specified by *position*. The UI\_POSITION structure is greater than *position* if the column value of the UI\_POSITION structure is greater than the column value of *position* or if the line value of the UI\_POSITION structure is greater than the line value of *position*.

- *returnValue<sub>out</sub>* is TRUE if the UI\_POSITION structure is greater than *position*. Otherwise, *returnValue* is FALSE.
- *position<sub>in</sub>* is the position to be compared.

## Example

```
#include <ui_dsp.hpp>

ExampleFunction ( )
{
    if (position > tPosition)
    {

    }
}
```

## UI\_POSITION::operator >=

### Syntax

```
#include <ui_dsp.hpp>

int operator >= (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 operator overload determines whether the `UI_POSITION` structure is greater than or equal to the `UI_POSITION` structure specified by *position*. The `UI_POSITION` structure is greater than or equal to *position* if the column value of the `UI_POSITION` structure is greater than or equal to the column value of *position* or if the line value of the `UI_POSITION` structure is greater than or

- *returnValue<sub>out</sub>* is TRUE if the `UI_POSITION` structure is greater than or equal to *position*. Otherwise, *returnValue* is FALSE.
- *position<sub>in</sub>* is the position to be compared.

## Example

```
#include <ui_dsp.hpp>

ExampleFunction ( )
{
    if (position >= tPosition)
    {

    }
}
```

## UI\_POSITION::operator <=

### Syntax

```
#include <ui_dsp.hpp>
```

```
int operator <= (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 operator overload determines whether the UI\_POSITION structure is less than or equal to the UI\_POSITION structure specified by *position*. The UI\_POSITION structure is less than or equal to *position* if the column value of the UI\_POSITION structure is less than or equal to the column value of *position* or if the line value of the UI\_POSITION structure is less than or equal to the line value of *position*.

- *returnValue<sub>out</sub>* is TRUE if the UI\_POSITION structure is less than or equal to *position*. Otherwise, *returnValue* is FALSE.
- *position<sub>in</sub>* is the position to be compared.

## Example

```
#include <ui_dsp.hpp>

ExampleFunction ( )
{
    if (position <= tPosition)
    {
        }
    }
}
```

## UI\_POSITION::operator ++

### Syntax

```
#include <ui_dsp.hpp>

UI_POSITION &operator ++ (void);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload increments the *line* and *column* values of the UI\_POSITION structure by one.

- *returnValue<sub>out</sub>* is a pointer to the UI\_POSITION structure after the value has been incremented. This pointer is returned so that the operator may be used in a statement containing other operations.

## Example

```
#include <ui_dsp.hpp>

ExampleFunction ( )
{
```

```
    position++;  
}
```

## UI\_POSITION::operator --

### Syntax

```
#include <ui_dsp.hpp>
```

```
UI_POSITION &operator -- (void);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload decrements the *line* and *column* values of the UI\_POSITION structure by one.

- *returnValue<sub>out</sub>* is a pointer to the UI\_POSITION structure after the position has been decremented. This pointer is returned so that the operator may be used in a statement containing other operations.

### Example

```
#include <ui_dsp.hpp>  
ExampleFunction (  
{  
  
    position--;  
}
```

## UI\_POSITION::operator +=

### Syntax

```
#include <ui_dsp.hpp>
```

```
UI_POSITION &operator += (int offset);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload increments the *line* and *column* values of the UI\_POSITION structure by *offset*.

- *returnValue<sub>out</sub>* is a pointer to the UI\_POSITION structure. This pointer is returned so that the operator may be used in a statement containing other operations.
- *offset<sub>in</sub>* is the value to be added to the position values.

### Example

```
#include <ui_dsp.hpp>
ExampleFunction ( )
{
    position +=5;
}
```

## UI\_POSITION::operator -=

### Syntax

```
#include <ui_dsp.hpp>
```

```
UI_POSITION &operator -= (int offset);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload decrements the *line* and *column* values of the UI\_POSITION structure by *offset*.

- *returnValue<sub>out</sub>* is a pointer to the UI\_POSITION structure. This pointer is returned so that the operator may be used in a statement containing other operations.
- *offset<sub>in</sub>* is the value to be subtracted from the position values.

### Example

```
#include <ui_dsp.hpp>
ExampleFunction ( )
{
    position -= 5;
}
```

## CHAPTER 32-UI\_PRINTER

The UI\_PRINTER class object is used to send output to a printer. In those environments that support printers directly (e.g., MS-Windows, OS/2, Macintosh, and NEXTSTEP) this class uses the operating system's API. So any printer supported by that environment is supported by OpenZinc Application Framework. In DOS, Epson®-compatible dot-matrix printers, Hewlett Packard PCL printers, and PostScript® printers are supported. In Motif PostScript printers are supported.

The printer class can be used either to draw custom images using display primitives or to simply dump a portion of the screen to the printer.

The display primitives are documented in "Chapter 7—UI\_DISPLAY" of this manual. Only those functions unique to the UI\_PRINTER class are documented here.

The UI\_PRINTER class is declared in **UI\_DSP.HPP**. Its public and protected members are:

```
Class ZIL_EXPORT_CLASS UI_PRINTER : public UI_DISPLAY
{
public:
    ZIL_PRINTER_MODE printerMode;
    UI_DISPLAY *display;

    Struct ZIL_EXPORT_CLASS POSTSCRIPTFONT
    {
        char *typeFace;
        short pointSize;
    };
    static POSTSCRIPTFONT psFontTable[ZIL_MAXFONTS];

#if defined(ZIL_MSDOS)
    int lPort;
#elif defined(ZIL_MSWINDOWS)
    HDC hdc;
    Static HFONT fontTable[ZIL_MAXFONTS] ;
#elif defined(ZIL_OS2)
    HDC hdc;
    static FONTMETRICS fontTable[ZIL_MAXFONTS];
#elif defined(ZIL_MACINTOSH)
    GrafPtr displayPort;
    TPPrPort printerPort;
    THPrint printJob;
    Struct PRINTERFONT
    {
        short font;
        Style face;
        short mode;
        short size;
        FontRec **fRec;
    };
    static PRINTERFONT fontTable[ZIL_MAXFONTS];
#elif defined(ZIL_NEXTSTEP)
    Struct NEXTFONT
    {
        id font;
    };
};
```



```

};
static NEXTFONT fontTable[ZIL_MAXFONTS];
#endif

UI_PRINTER(UI_DISPLAY *display = ZIL_NULLP(UI_DISPLAY));
virtual ~UI_PRINTER(void);
virtual void Bitmap(ZIL_SCREENID screenID, int column, int line,
    int bitmapWidth, int bitmapHeight, const ZIL_UINT8 *bitmapArray,
    const UI_PALETTE *palette = ZIL_NULLP(UI_PALETTE),
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
    ZIL_BITMAP_HANDLE *colorBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE),
    ZIL_BITMAP_HANDLE *monoBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE));
virtual void Ellipse(ZIL_SCREENID screenID, int column, int line,
    int startAngle, int endAngle, int xRadius, int yRadius,
    const UI_PALETTE *palette, int fill = FALSE, int xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void Line(ZIL_SCREENID screenID, int column1, int line1,
    int column2, int line2, const UI_PALETTE *palette, int width = 1,
    int xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual ZIL_COLOR MapColor(const UI_PALETTE *palette, int isForeground);
virtual void Polygon(ZIL_SCREENID screenID, int numPoints,
    const int *polygonPoints, const UI_PALETTE *palette,
    int fill = FALSE, int xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void Rectangle(ZIL_SCREENID screenID, const UI_REGION &region,
    const UI_PALETTE *palette, int width = 1, int fill = FALSE,
    int xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void Rectangle(ZIL_SCREENID screenID, int left, int top,
    int right, int bottom, const UI_PALETTE *palette, int width = 1,
    int fill = FALSE, int xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void Text(ZIL_SCREENID screenID, int left, int top,
    const ZIL_ICHAR *text, const UI_PALETTE *palette, int length = -1,
    int fill = TRUE, int xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextHeight(const ZIL_ICHAR *string,
    ZIL_SCREENID screenID = ID_SCREEN,
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextWidth(const ZIL_ICHAR *string,
    ZIL_SCREENID screenID = ID_SCREEN,
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int VirtualGet(ZIL_SCREENID screenID, const UI_REGION &region);
virtual int VirtualGet(ZIL_SCREENID screenID, int left, int top,
    int right, int bottom);
virtual int VirtualPut(ZIL_SCREENID screenID);

// New routines for Printer.
virtual int BeginPrintJob(ZIL_PRINTER_MODE pMode = PRM_DEFAULT,
    char *fileName = ZIL_NULLP(char));
virtual void EndPrintJob(void);
virtual void BeginPage(void);
virtual void EndPage(void);
virtual void ScreenDump(ZIL_SCREENID screenID, UI_REGION kregion,
    ZIL_PRINTER_MODE = PRM_DEFAULT, char *fileName = ZIL_NULLP(char));

```

## General Members

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

- *printerMode* indicates what printer mode the device is in. *printerMode* can have one of the following values:

**PRM\_DEFAULT**—Causes the device to attempt to connect directly to the default printer.

**PRM\_DOTMATRIX9**—Causes the device to print to a 9-pin Epson-compatible dot matrix printer. This mode is specific to DOS.

**PRM\_DOTMATRIX24**—Causes the device to print to a 24-pin Epson-compatible dot matrix printer. This mode is specific to DOS.

**PRM\_LASER**—Causes the device to print to a Hewlett Packard PCL printer. This mode is specific to DOS.

**PRM\_POSTSCRIPT**—Causes the device to print to a PostScript file.

- *display* is a pointer to the display. A valid pointer for the display used by the application must be supplied if screen dumps are to be used.
- *POSTSCRIPTFONT* identifies PostScript fonts.

*typeFace* is the name of the font.

*pointSize* is the point size of the font.

- *psFontTable* is a table of PostScript fonts used by the printer. Initially, this table has ten common PostScript fonts in it.
- *IPort* is the line port being printed to. This member is available in DOS only.
- *hdc* is the device context handle for the printer. This member is available in Windows and OS/2 only.
- *fontTable* corresponds to the font table used by the display class.
- *displayPort* is the current display port. It is saved so that it can be properly restored when printing is completed. This member is available in Macintosh only.
- *printerPort* is the printer port being printed to. This member is available in Macintosh only.

- *printJob* is information about the print job that was set up by the end-user.
- *PRINTERFONT* is a structure that identifies Macintosh fonts. It corresponds to the display class font structure.
- *NEXTFONT* is a structure that identifies NEXTSTEP fonts. It corresponds to the display class font structure.

## **UI\_PRINTER::~UI\_PRINTER**

### **Syntax**

```
static #include <ui_dsp.hpp>

UI_PRINTER(UI_DISPLAY *display = ZIL_NULLP(UI_DISPLAY));
```

### **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 `UI_PRINTER` class object.

- *display<sub>in</sub>* is a pointer to the display. A valid display pointer must be provided if screen dumps are required.

## **UI\_PRINTER::~~UI\_PRINTER**

### **Syntax**

```
static #include <ui_dsp.hpp>

~UI_PRINTER(void);
```

## Portability

This function is available on the following environments:

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

## Remarks

This destructor destroys the UI\_PRINTER class information.

## UI\_PRINTER::BeginPage

### Syntax

```
#include <ui_dsp.hpp>
```

```
virtual void BeginPage(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 initializes the device to begin accepting commands for a new page in the print job. Any calls to the printer primitives made after a call to this function will be buffered until a call to **EndPage()** is made. When **EndPage()** is called, the page will be printed. **BeginPage()** must be called before any printer primitives commands are generated.

## UI\_PRINTER::BeginPrintJob

### Syntax

```
#include <ui_dsp.hpp>

virtual int BeginPrintJob(ZIL_PRINTER_MODE pMode = PRM_DEFAULT,
    char * _fileName = ZIL_NULLP(char));
```

### 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 begins the printing process by initializing any required information and setting up the printer. Printing can be sent to a file on disk if desired. This function must be called at the beginning of the entire print job.

- *returnValue<sub>out</sub>* indicates the success of the function call. *returnValue* is FALSE if the function was unsuccessful. Otherwise, it is TRUE.
- *pMode<sub>in</sub>* indicates the printer mode desired. See the description of *printerMode*, above, for the possible modes.
- *\_fileName<sub>in</sub>* is the name of the file to print to if PostScript output should go to a file. If no filename is provided, output will go directly to the printer.

In Motif, if *pMode* is PRM\_DEFAULT, thus causing the device to output to the printer directly, then *\_fileName* is used to specify commands for the print job. If *\_fileName* is NULL, the print job will be piped to "lpr" by default. If other options are desired, such as specifying the name of the printer and how many copies should be printed, this string should be set accordingly. For example, if the printer name is PostScriptPrinter and 3 copies are desired, *\_fileName* should be "lpr -PPostScriptPrinter -#3."

## UI\_PRINTER::EndPage

### Syntax

```
#include <ui_dsp.hpp>

virtual void EndPage(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 prints the current page. After calling **BeginPage()**, all printer primitive functions will be buffered. When an image is completed, **EndPage()** should be called to send the page to the printer. **EndPage()** clears the page, so subsequent calls to printer primitives will appear on a new page.

## UI\_PRINTER::EndPrintJob

### Syntax

```
#include <ui_dsp.hpp>

virtual void EndPrintJob(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 ends the printing process by sending any necessary escape sequences to the printer. This function must be called at the end of the entire print job.

## UI\_PRINTER::ScreenDump

### Syntax

```
#include <ui_dsp.hpp>

virtual void ScreenDump(ZIL_SCREENID screenID, UI_REGION &region,
    ZIL_PRINTER_MODE pMode = PRM_DEFAULT,
    char * _fileName = ZIL_NULLP(char));
```

### 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 dumps a portion of the screen to the printer. This function calls **BeginPrintJob()**, **BeginPage()**, **EndPage()**, and **EndPrintJob()**.

- *screenID<sub>in</sub>* is the screenID of the window being printed.
- *region<sub>in</sub>* is the region of the screen that is to be printed, *region* is relative to the upper-left corner of the window identified by *screenID* and is in screen coordinates. Thus, if the display is a text mode display, *region* is in cell coordinates. Otherwise, it is in pixel coordinates.
- *pMode<sub>in</sub>* indicates the printer mode desired. See the description of *printerMode*, above, for the possible modes.
- *\_fileName<sub>in</sub>* is the name of the file to print to if PostScript output should go to a file. If no filename is provided, output will go directly to the printer.

In Motif, if *pMode* is PRM\_DEFAULT, thus causing the device to output to the printer directly, then *\_fileName* is used to specify commands for the print job. If *\_fileName* is NULL, the print job will be piped to "lpr" by default. If other options are desired, such as specifying the name of the printer and how many copies should be printed, this string should be set accordingly. For example, if the printer name is PostScriptPrinter and 3 copies are desired, *\_fileName* should be "lpr -PPostScript-Printer -#3."





## CHAPTER 33 - UI\_QUEUE\_BLOCK

The UI\_QUEUE\_BLOCK is an advanced class that is only used by the Event Manager. In general, programmers should not be concerned with it. The UI\_QUEUE\_BLOCK class is an array of UI\_QUEUE\_ELEMENT objects that acts like a doubly-linked list. Because it is an array that is created at the beginning of the program, manipulating the list is much faster than if memory is allocated and deallocated each time an element is added or subtracted in the list. See "Chapter 21—UI\_LIST\_BLOCK" for more details about the operation of a list block.

The UI\_QUEUE\_BLOCK class is declared in **UI\_EVT.HPP**. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UI_QUEUE_BLOCK : public UI_LIST_BLOCK
{
public:
    UI_QUEUE_BLOCK(int noOfElements);
    ~UI_QUEUE_BLOCK(void);

    UI_QUEUE_ELEMENT *Current(void);
    UI_QUEUE_ELEMENT *First(void);
    UI_QUEUE_ELEMENT *Last(void);
};
```

### General Members

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

### UI\_QUEUE\_BLOCK::UI\_QUEUE\_BLOCK

#### Syntax

```
#include <ui_evt.hpp>
```

```
UI_QUEUE_BLOCK(int noOfElements)\
```

## Portability

This function is available on the following environments:

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

## Remarks

This constructor allocates an array of `UI_QUEUE_ELEMENT` classes and initializes them to behave like a list. This is done so that the programmer can have access to all of the list functions without having to create new array functions.

- `noOfElementsin` designates the number of elements to be assigned space in memory.

## Example

```
#include <ui_evt.hpp>

UI_QUEUE_BLOCK::UI_QUEUE_BLOCK(int _noOfElements) :
    UI_LIST_BLOCK(_noOfElements)
{
    // Initialize the queue block.
    UI_QUEUE_ELEMENT *queueBlock = new UI_QUEUE_ELEMENT[_noOfElements];
    elementArray = queueBlock;
    for (int i = 0; i < _noOfElements; i++)
        freeList.Add(NULL, kqueueBlock[i]);
}

UI_EVENT_MANAGER::UI_EVENT_MANAGER(UI_DISPLAY *display, int _noOfElements) :
    UI_LIST(UI_DEVICE::CompareDevices), queueBlock(_noOfElements), level(1)
{
    display = _display;
    UI_DEVICE::display = display;
    UI_DEVICE::eventManager = this;
}
```

## UI\_QUEUE\_BLOCK: fUI\_QUEUE\_BLOCK

### Syntax

```
#include <ui_evt.hpp>

~UI_QUEUE_BLOCK(void);
```

## Portability

This function is available on the following environments:

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

## Remarks

This destructor destroys the class information associated with the UI\_QUEUE\_BLOCK class. It also destroys each element in the queue block.

## Example

```
#include <ui_evt.hpp>

UI_QUEUE_BLOCK::~UI_QUEUE_BLOCK(void)
{
    // Free the queue block.
    UI_QUEUE_ELEMENT *queueBlock = (UI_QUEUE_ELEMENT *)elementArray;
    delete [noOfElements]queueBlock;
}
```

## UI\_QUEUE\_BLOCK::Current

### Syntax

```
#include <ui_gen.hpp>

UI_QUEUE_ELEMENT *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 element, if one exists, in the list.

- *returnValue<sub>out</sub>* is a pointer to the current element in the list. If there is no current element, *returnValue* is NULL.

## UI\_QUEUE\_BLOCK::First

### Syntax

```
#include <ui_gen.hpp>

UI_QUEUE_ELEMENT *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 element, if one exists, in the list.

- *returnValue<sub>out</sub>* is a pointer to the first element in the list. If there is no first element, *returnValue* is NULL.

## UI\_QUEUE\_BLOCK::Last

### Syntax

```
#include <ui_gen.hpp>

UI_QUEUE_ELEMENT *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 element, if one exists, in the list.

- *returnValue<sub>out</sub>* is a pointer to the last element in the list. If there is no last element, *returnValue* is NULL.



## **CHAPTER 34 - UI QUEUE ELEMENT**

The UI\_QUEUE\_ELEMENT class is an advanced class that is only used by the UI\_QUEUE\_BLOCK class within the Event Manager. It contains the UI\_EVENT structure, which contains an event to be processed by the system. In general, programmers should not be concerned with it.

The UI\_QUEUE\_ELEMENT class is declared in **UI\_EVT.HPP**. Its public and protected members are:

```
Class ZIL_EXPORT_CLASS UI_QUEUE_ELEMENT : public UI_ELEMENT
{
public:
    UI_QUEUE_ELEMENT(void);
    ~UI_QUEUE_ELEMENT(void);
    UI_EVENT event;

    UI_QUEUE_ELEMENT *Next(void);
    UI_QUEUE_ELEMENT *Previous(void);
};
```

### **General Members**

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

- *event* is the UI\_EVENT information associated with the queue element.

### **UI\_QUEUE\_ELEMENT::UI\_QUEUE\_ELEMENT**

#### **Syntax**

```
#include <ui_evt.hpp>
```

```
UI_QUEUE_ELEMENT(void);
```

#### **Portability**

This function is available on the following environments:

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



## Remarks

This constructor creates a UI\_QUEUE\_ELEMENT object.

## Example

```
#include <ui_evt.hpp>
UI_QUEUE_BLOCK::UI_QUEUE_BLOCK(int _noOfElements) :
    UI_LIST_BLOCK(_noOfElements)
{
    // Initialize the queue block.
    UI_QUEUE_ELEMENT *queueBlock = new UI_QUEUE_ELEMENT[_noOfElements];
    elementArray = queueBlock;
    for (int i = 0; i < _noOfElements; i++)
        freeList.Add(NULL, &queueBlock[i]);
}
```

## UI\_QUEUE\_ELEMENT::~~UI\_QUEUE\_ELEMENT

### Syntax

```
#include <ui_evt.hpp>

virtual ~UI_QUEUE_ELEMENT(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 UI\_QUEUE\_ELEMENT object. Care should be taken to only destroy a UI\_QUEUE\_ELEMENT class that is not attached to another associated object.

## UI\_QUEUE\_ELEMENT::Next

### Syntax

```
#include <ui_evt.hpp>
```

```
UI_QUEUE_ELEMENT *Next(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 next UI\_QUEUE\_ELEMENT in the list.

- *returnValue<sub>out</sub>* is a pointer to the next UI\_QUEUE\_ELEMENT in the list, if one exists. If one does not exist, *returnValue* will be NULL.

## UI\_QUEUE\_ELEMENT::Previous

### Syntax

```
#include <ui_evt.hpp>
```

```
UI_QUEUE_ELEMENT *Previous(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 previous `UI_QUEUE_ELEMENT` in the list.

- *returnValue*<sub>out</sub> is a pointer to the previous `UI_QUEUE_ELEMENT` in the list, if one exists. If one does not exist, *returnValue* will be `NULL`.

## CHAPTER 35 - UI REGION

The UI\_REGION structure is used to store and manipulate region information. A region is a rectangular area defined by its four corners. The UI\_REGION structure is typically used to define an object's screen location.

The UI\_REGION structure is declared in UI\_DSP.HPP. Its public and protected members are:

```
struct ZIL_EXPORT_CLASS UI_REGION
{
public:
    int left, top, right, bottom;

    #if defined(ZIL_MSWINDOWS)
        void Assign(const RECT &rect);
    #elif defined(ZIL_OS2)
        void Assign(const RECTL &rect);
    #elif defined(ZIL_MACINTOSH)
        void Assign(const Rect &rect);
    #elif defined(ZIL_NEXTSTEP)
        void Assign(const NXRect &rect);
    #endif
    int Encompassed(const UI_REGION &region) const;
    int Height(void) const;
    int Overlap(const UI_REGION &region) const;
    int Overlap(const UI_POSITION &position) const;
    int Touching(const UI_POSITION &position) const;
    int Overlap(const UI_REGION &region, UI_REGION &result) const;
    int Width(void) const;

    int operator==(const UI_REGION &region) const;
    int operator!=(const UI_REGION &region) const;
    UI_REGION &operator++(void);
    UI_REGION ^operator--(void);
    UI_REGION &operator+=(int offset);
    UI_REGION &operator-=(int offset);
};
```

### General Members

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

- *left* and *top* define the top-left corner of the region. These values may be in cells, minicells or pixels depending on the context of the UI\_REGION being used.
- *right* and *bottom* define the bottom-right corner of the region. These values may be in cells, minicells or pixels depending on the context of the UI\_REGION being used.

## UI\_REGION::Assign

### Syntax

```
#include <ui_dsp.hpp>

void Assign(const RECT &rect);
    or
void Assign(const RECTL &rect);
    or
void Assign(const Rect &rect);
    or
void Assign(const NXRect &rect);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function copies the region information from the operating system-specific structure into the UI\_REGION structure.

- *rect<sub>in</sub>* is the operating system-specific region structure whose value is to be copied into the UI\_REGION structure.

### Example

```
#include <ui_dsp.hpp>

ExampleFunction()
{
    region.Assign(tRegion);
}

}
```

## UI\_REGION-.Encompassed

### Syntax

```
#include <ui_dsp.hpp>

int Encompassed(const UI_REGION &region);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function determines if the UI\_REGION object is completely encompassed by the UI\_REGION structure specified by *region*.

- *returnValue<sub>out</sub>* is TRUE if the UI\_REGION object is encompassed by *region*. Otherwise, *returnValue* is FALSE.
- *region<sub>in</sub>* is the region to be compared.

### Example

```
#include <ui_dsp.hpp>

ExampleFunction ( )
{
    if (region1.Encompassed(region2))
    {

    }
}
```

## UI\_REGION::Height

### Syntax

```
#include <ui_dsp.hpp>

int Height(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 height of the region.

- *returnValue<sub>out</sub>* is the height of the region.

## UI\_REGION::Overlap

### Syntax

```
#include <ui_dsp.hpp>

int Overlap(const UI_REGION &region);
    or
int Overlap(const UI_POSITION &position);
    or

int Overlap(const UI_REGION &region, UI_REGION &result)\
```

### Portability

These functions are available on the following environments:

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

## Remarks

These overloaded functions determine if the UI\_REGION object overlaps another region or position.

The first overloaded function determines if the UI\_REGION object is overlapped by another UI\_REGION structure specified by *region*.

- *returnValue<sub>out</sub>* is TRUE if the UI\_REGION object is overlapped by *region*. Otherwise, *returnValue* is FALSE.
- *region<sub>m</sub>* is the region to be compared.

The second overloaded function determines if the UI\_REGION object is overlapped by a UI\_POSITION structure specified by *position*.

- *returnValue<sub>out</sub>* is TRUE if the UI\_REGION object is overlapped by *position*. Otherwise, *returnValue* is FALSE.
- *position<sub>in</sub>* is the position to be compared.

The third overloaded function tests to see if the UI\_REGION object is overlapped by the UI\_REGION structure specified by *region*. The overlapping portion of the regions is copied into *result*.

- *returnValue<sub>out</sub>* is TRUE if the UI\_REGION object is overlapped by *region*. Otherwise, *returnValue* is FALSE.
- *region<sub>in</sub>* is the region to be compared.
- *result<sub>out</sub>* is the region overlapped by both the UI\_REGION object and *region*.

## Example 1

```
#include <ui_dsp.hpp>
EVENT_TYPE UID_CURSOR::Event(const UI_EVENT &event)
{
    switch (event.type)
```



```

case D_SHOW:
    if (state != D_OFF)
    {
        UI_REGION region;
        region.left = position.column;
        region.top = position.line;
        region.right = region.left + display->cellwidth - 1;
        region.bottom = region.top + display->cellHeight - 1;
        if (region.Overlap(event.region))
            tState = (event.rawCode == D_HIDE) ? D_HIDE : D_ON;
    }
    break;
}
}

```

## Example 2

```

#include <ui_win.hpp>

EVENT_TYPE UIW_VT_LIST::Event(const UI_EVENT &event)
{
    if (object && (ccode == L_SELECT || trueRegion.Overlap(event.position)))
        object->Event(UI_EVENT(L_SELECT));
    break;

    // Return the control code,
    return (ccode);
}

```

## UI\_REGION::Touching

### Syntax

```

#include <ui_dsp.hpp>

int Touching(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 determines whether the point specified by *position* is touching the edge of the UI\_REGION object, *position* is considered to be touching the region if the point defined by *position* is exactly on an edge defined by the UI\_REGION structure.

- *returnValue<sub>out</sub>* is TRUE if *position* is touching the edge of the UI\_REGION object. Otherwise, *returnValue* is FALSE.
- *position<sub>in</sub>* is the position to be compared.

## UI\_REGION::Width

### Syntax

```
#include <ui_dsp.hpp>

int Width(void);
```

### Portability

This function is available on the following environments:

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

## Remarks

This function returns the width of the region.

- *returnValue<sub>out</sub>* is the width of the region.

## UI\_REGION::**operator ==**

### Syntax

```
#include <ui_dsp.hpp>

int operator == (const UI_REGION &region);
```

## Portability

This function is available on the following environments:

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

## Remarks

This operator overload determines if the UI\_REGION object has the same region as the UI\_REGION specified by *region*.

- *returnValue<sub>out</sub>* is TRUE if the UI\_REGION is the same as *region*. Otherwise, *returnValue* is FALSE.
- *region<sub>in</sub>* is the region to be compared.

## Example

```
#include <ui_dsp.hpp>
ExampleFunction ( )
    if (region1 == region2)
```

## UI\_REGION::operator !=

### Syntax

```
#include <ui_dsp.hpp>
int operator != (const UI_REGION &region)\
```

## Portability

This function is available on the following environments:

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

## Remarks

This operator overload determines if the `UI_REGION` object does not have the same region as the `UI_REGION` object specified by *region*.

- *returnValue<sub>out</sub>* is `TRUE` if `UI_REGION` is not the same as *region*. Otherwise, *returnValue* is `FALSE`.
- *region<sub>m</sub>* is the region to be compared.

## Example

```
#include <ui_dsp.hpp>
ExampleFunction ( )
{
    if (region1 != region2)
    {

    }
}
```

## UI\_REGION::operator ++

### Syntax

```
#include <ui_dsp.hpp>
UI_REGION &operator ++ (void);
```

## Portability

This function is available on the following environments:

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

## Remarks

This operator overload increases the size of the region on each side by one. It does this by decrementing the *left* and *top* values of the region by one and incrementing the *right* and *bottom* values of the region by one.

- *returnValue<sub>out</sub>* is a pointer to the UI\_REGION object after its size has been modified. This pointer is returned so that the operator may be used in a statement containing other operations.

## Example

```
#include <ui_dsp.hpp>
ExampleFunction ( )
{
    ++region;
}
```

## UI\_REGION-operator --

### Syntax

```
#include <ui_dsp.hpp>

UI_REGION &operator -- (void);
```

## Portability

This function is available on the following environments:

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

## Remarks

This operator overload decreases the size of the region on each side by one. It does this by incrementing the *left* and *top* values of the region by one and decrementing the *right* and *bottom* values of the region by one.

- *returnValue<sub>out</sub>* is the address of the UI\_REGION object after its size has been modified. This pointer is returned so that the operator may be used in a statement containing other operations.

## Example

```
#include <ui_dsp.hpp>

ExampleFunction ( )
{

    --region;

}
```

## UI\_REGION::operator +=

### Syntax

```
#include <ui_dsp.hpp>

UI_REGION &operator += (int offset);
```

### Portability

This function is available on the following environments:

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

## Remarks

This operator overload increases the size of the region on each side by *offset*. It does this by subtracting *offset* from the *left* and *top* values of the region and adding *offset* to the *right* and *bottom* values of the region.

- *returnValue<sub>out</sub>* is a pointer to the UI\_REGION object after its size has been modified. This pointer is returned so that the operator may be used in a statement containing other operations.

## Example

```
#include <ui_dsp.hpp>
ExampleFunction ( )
{
    region += 5;
}
```

## UI\_REGION: operator -=

### Syntax

```
#include <ui_dsp.hpp>

UI_REGION &operator -= (int offset);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload decreases the size of the region on each side by *offset*. It does this by adding *offset* to the *left* and *top* values of the region and subtracting *offset* from the *right* and *bottom* values of the region.

- *returnValue<sub>out</sub>* is a pointer to the UI\_REGION object after its size has been modified. This pointer is returned so that the operator may be used in a statement containing other operations.

## Example

```
#include <ui_dsp.hpp>
ExampleFunction ( )

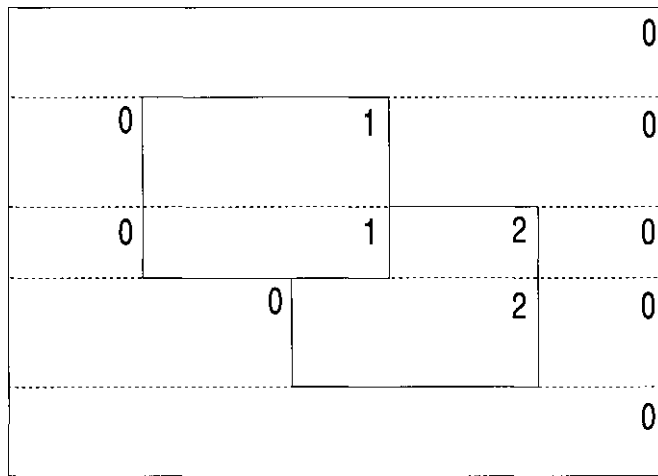
    region -= 5;
}
```





## CHAPTER 36-UI\_REGION\_ELEMENT

The `UI_REGION_ELEMENT` class works with the `UI_REGION_LIST` class to maintain a list of rectangular screen regions. The screen regions are used to calculate a window's available region and to perform clipping more efficiently when updating the display. When an object with the `WOF_NON_FIELD_REGION` flag set is added to a window, the window's available region is updated to prevent allocating that space in the future. The window's available region is maintained by a `UI_REGION_LIST`. Also, whenever a window is placed on the screen or an existing window's position or size is changed, the affected areas of the display must be updated. The screen's regions are maintained in a `UI_REGION_LIST` that allows OpenZinc to efficiently update the display. The picture below shows how a screen may be split up (where 0 is the screen background and 1 and 2 are overlapping windows):



In general, the programmer does not need to use this class.

The `UI_REGION_ELEMENT` class is declared in `UI_DSP.HPP`. Its public and protected members are:

```
Class ZIL_EXPORT_CLASS UI_REGION_ELEMENT : public UI_ELEMENT
{
public:
    ZIL_SCREENID screenID;
    UI_REGION region;

    UI_REGION_ELEMENT(ZIL_SCREENID screenID, const UI_REGION &region);
    UI_REGION_ELEMENT(ZIL_SCREENID screenID, int left, int top, int right,
        int bottom);
    ~UI_REGION_ELEMENT(void);
};
```

```
// Element members.
UI_REGION_ELEMENT *Next(void);
UI_REGION_ELEMENT *Previous(void);
```

## General Members

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

- *screenID* identifies which object "owns" the region. *screenID* is an identifier associated with a window object. See the *screenID* section of "Chapter 43—UI\_WINDOW\_OBJECT" of this manual and "Appendix A—Support Definitions" of *Programmer's Reference Volume 2* for more information.
- *region* is the rectangular region that is reserved.

## UI\_REGION\_ELEMENT::UI\_REGION\_ELEMENT

### Syntax

```
#include <ui_dsp.hpp>
```

```
UI_REGION_ELEMENT(ZIL_SCREENID screenID, const UI_REGION &region);
```

*or*

```
UI_REGION_ELEMENT(ZIL_SCREENID screenID, int _left, int _Top, int _right,  
int _bottom);
```

### Portability

These functions are available on the following environments:

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

### Remarks

These overloaded constructors create a new UI\_REGION\_ELEMENT object.

- *\_screenID* is the identification to associate with the region.

- `_regionin` is the region to define.
- `_leftin`, `_topin`, `_rightin`, and `_bottomin` is the region to define.

## Example

```
#include <ui_dsp.hpp>

void UI_DISPLAY::RegionDefine(ZIL_SCREENID screenID, int left, int top,
                             int right, int bottom)
{
    UI_REGION region;
    region.left = left;
    region.top = top;
    region.right = right;
    region.bottom = bottom;
    // See if it is a full screen definition,
    if (region.left <= 0 && region.top <= 0 &&
        region.right >= columns - 1 && region.bottom >= lines - 1)
    {
        UI_REGION_LIST::Destroy();
        Add(0, new UI_REGION_ELEMENT(screenID, 0, 0, columns - 1, lines - UK-
        return;
    }
}
```

## UI\_REGION\_ELEMENT::~~UI\_REGION\_ELEMENT

### Syntax

```
#include <ui_dsp.hpp>

~UI_REGION_ELEMENT(void);
```

### Portability

This function is available on the following environments:

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

## Remarks

This destructor destroys the class information associated with the UI\_REGION\_ELEMENT object.

## Example

```
#include <ui_dsp.hpp>

void UI_REGION_LIST::Split(int screenID, const UI_REGION &region)
{
    UI_REGION tRegion, sRegion;
    UI_REGION_ELEMENT *dRegion, *t_dRegion;
    // Split any overlapping regions.
    for (dRegion = First(); dRegion; dRegion = t_dRegion)
    {
        // Region 3 is the object's region.
        UI_LIST::Subtract(dRegion);
        delete dRegion;
    }
}
```

## UI\_REGION\_ELEMENT::Next

### Syntax

```
#include <ui_dsp.hpp>

UI_REGION_ELEMENT *Next(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 next UI\_REGION\_ELEMENT in the list.

- *returnValue<sub>out</sub>* is a pointer to the next UI\_REGION\_ELEMENT in the list, if one exists. If one does not exist, *returnValue* will be NULL.

## UI\_REGION\_ELEMENT::Previous

### Syntax

```
#include <ui_dsp.hpp>

UI_REGION_ELEMENT *Previous(void);
```

### Portability

This function is available on the following environments:

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

### Remarks

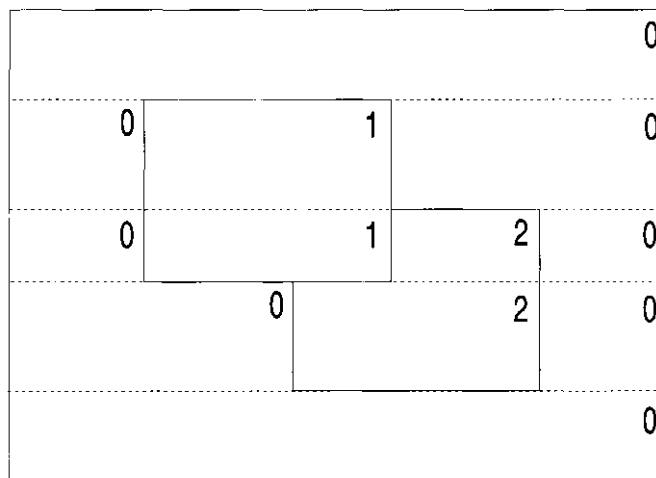
This function returns a pointer to the previous UI\_REGION\_ELEMENT in the list.

- *returnValue<sub>out</sub>* is a pointer to the previous UI\_REGION\_ELEMENT in the list, if one exists. If one does not exist, *returnValue* will be NULL.



## CHAPTER 37-UI\_REGION\_LIST

The `UI_REGION_LIST` class works with the `UI_REGION_ELEMENT` class to maintain a list of rectangular screen regions. The screen regions are used to calculate a window's available region and to perform clipping more efficiently when updating the display. When an object with the `WOF_NON_FIELD_REGION` flag set is added to a window, the window's available region is updated to prevent allocating that space in the future. The window's available region is maintained by a `UI_REGION_LIST`. Also, whenever a window is placed on the screen or an existing window's position or size is changed, the affected areas of the display must be updated. The screen's regions are maintained in a `UI_REGION_LIST` that allows OpenZinc to efficiently update the display. The picture below shows how a screen may be split up (where 0 is the screen background and 1 and 2 are overlapping windows):



The `UI_REGION_LIST` class is declared in `UI_DSP.HPP`. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UI_REGION_LIST : public UI_LIST
public:
    UI_REGION_LIST(void);
    ~UI_REGION_LIST(void);
    void Split(ZIL_SCREENID screenID, const UI_REGION &region,
              int allocateBelow = FALSE);

    // List members.
    UI_REGION_ELEMENT *Current(void);
    UI_REGION_ELEMENT *First(void);
```



```
    UI_REGION_ELEMENT *Last(void);  
};
```

## General Members

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

### UI\_REGION\_LIST::Current

#### Syntax

```
#include <ui_dsp.hpp>
```

```
UI_REGION_ELEMENT *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 element, if one exists, in the list.

- *returnValue<sub>out</sub>* is a pointer to the current element in the list. If there is no current element, *returnValue* is NULL.

### UI\_REGION\_LIST::First

#### Syntax

```
#include <ui_dsp.hpp>
```

```
UI_REGION_ELEMENT *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 element, if one exists, in the list.

- *returnValue<sub>out</sub>* is a pointer to the first element in the list. If there is no first element, *returnValue* is NULL.

## UI\_REGION\_LIST::Last

### Syntax

```
#include <ui_dsp.hpp>
```

```
UI_REGION_ELEMENT *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 element, if one exists, in the list.

- *returnValue<sub>out</sub>* is a pointer to the last element in the list. If there is no last element, *returnValue* is NULL.

## UI\_REGION\_LIST::Split

### Syntax

```
#include <ui_dsp.hpp>

void Split(ZIL_SCREENID screenID, const UI_REGION &region,
           int allocateBelow = 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 splits the regions in the region list and adds the new region to the list.

- *screenID*<sub>in</sub> is the identification to associate with the new region.
- *region*<sub>in</sub> is the new region to define.
- *allocateBelow*<sub>in</sub> specifies whether a new region should be created if there are no regions in the list.

### Example

```
#include <ui_dsp.hpp>

void UI_DISPLAY::RegionDefine(ZIL_SCREENID screenID, int left, int top, int
right,
int bottom)
{
    UI_REGION region;
    region.left = left;
    region.top = top;
    region.right = right;
    region.bottom = bottom;

    // Clip regions partially off the screen to fit on the screen,
    if (region.left < 0)
        region.left = 0;
    if (region.right >= columns)
```

```
        region.right = columns - 1;
    if (region.top < 0)
        region.top = 0;

    if (region.bottom >= lines)
        region.bottom = lines - 1;
    // Split any overlapping regions.
    Split(screenID, region);

    // Define the new display region.
    Add(0, new UI_REGION_ELEMENT(screenID, kregion));
}
```



## CHAPTER 38 - UI\_RELATIVE\_CONSTRAINT

The `UI_RELATIVE_CONSTRAINT` class object is used for geometry management. Specifically, this class allows a managed object to be tied to an edge of its parent at a distance relative to the size of the parent. For example, a button can be positioned so that its left edge is always twenty-five percent of the way across its parent, even if the parent is sized. The `UI_RELATIVE_CONSTRAINT` is added to the parent object's geometry manager. See "Chapter 14—`UI_GEOMETRY_MANAGER`" for more details on using the geometry manager.

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

```
Class UI_RELATIVE_CONSTRAINT : public UI_CONSTRAINT
{
public:
    UI_RELATIVE_CONSTRAINT(UI_WINDOW_OBJECT *_object,
        RLCF_FLAGS_rlcFlags = RLCF_NO_FLAGS,
        int_numerator = 50, int_denominator = 100, int_offset = 0);
    virtual ~UI_RELATIVE_CONSTRAINT(void);

    virtual void *Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID ObjectID = ID_DEFAULT);
    virtual void Modify(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));
    UI_RELATIVE_CONSTRAINT(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 numerator;
    int denominator;
    RLCF_FLAGS_rlcFlags;
    int offset;
};
```

## General Members

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

- *numerator* is used in conjunction with *denominator* to determine the relative positioning of the managed object, *numerator* is divided by *denominator* to obtain a percentage. The managed object will be positioned according to this percentage of the parent window's size.
- *denominator* is used in conjunction with *numerator* to determine the relative positioning of the managed object, *numerator* is divided by *denominator* to obtain a percentage. The managed object will be positioned according to this percentage of the parent window's size.
- *rlcFlags* are flags that define the operation of the UI\_RELATIVE\_CONSTRAINT class. A full description of the relative constraint flags is given in the UI\_RELATIVE\_CONSTRAINT constructor.
- *offset* is how far, in addition to the percentage determined using *numerator* and *denominator*, the managed object should be positioned from the object to which it is tied, *offset* number of cells are added to the position determined by the percentage. This value is specified in cell dimensions.

## UI\_RELATIVE\_CONSTRAINT::UI\_RELATIVE\_CONSTRAINT

### Syntax

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

```
UI_RELATIVE_CONSTRAINT(UI_WINDOW_OBJECT *_object,  
    RLCF_FLAGS _rlcFlags = RLCF_NO_FLAGS, int _numerator = 50,  
    int _denominator = 100, int _offset = 0);
```

### 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 `UI_RELATIVE_CONSTRAINT` object.

- `_objectin` is the object to be managed.
- `_rlcFlagsin` are flags that define the operation of the `UI_RELATIVE_CONSTRAINT` class. The following flags (declared in `UI_WIN.HPP`) control the general operation of a `UI_RELATIVE_CONSTRAINT` class object:

`RLCF_BOTTOM`—Maintains the bottom edge of the managed object at the specified relative distance from the object to which it is tied.

`RLCFJLEFT`—Maintains the left edge of the managed object at the specified relative distance from the object to which it is tied.

`RLCF_HORIZONTAL_CENTER`—Causes the object to be centered horizontally within its parent.

`RLCF_OPPOSITE`—Causes the managed object to be tied to the opposite edge of the object to which it is tied. For example, if the `RLCF_TOP` flag is set, the top edge of the managed object will be tied to the bottom edge of the object to which it is tied.

`RLCF_NO_FLAGS`—Does not associate any special flags with the `UI_RELATIVE_CONSTRAINT` class. This flag should not be used in conjunction with any other `RLCF` flags.

`RLCF_RIGHT`—Maintains the right edge of the managed object at the specified relative distance from the object to which it is tied.

`RLCF_STRETCH`—Causes the managed object to be stretched, if necessary, to maintain its constraints. For example, if the left and right edges of the object are tied to the parent window and the window is sized, the managed object must stretch or shrink to maintain its distance from the edges.

`RLCF_TOP`—Maintains the top edge of the managed object at the specified relative distance from the object to which it is tied.

`RLCF_VERTICAL_CENTER`—Causes the object to be centered vertically within its parent.

- `_numeratorin` is divided by `denominator` to obtain the relative positioning constraint.



- *\_denominator<sub>in</sub>* is divided into *numerator* to obtain the relative positioning constraint.
- *\_offset<sub>in</sub>* is an additional fixed distance at which the managed object will be maintained.

## **UI\_RELATIVE\_CONSTRAINT::~~UI\_RELATIVE\_CONSTRAINT**

### **Syntax**

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

```
virtual ~UI_RELATIVE_CONSTRAINT(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 `UI_RELATIVE_CONSTRAINT` object.

## **UI\_RELATIVE\_CONSTRAINT::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<sub>out</sub>* 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<sub>in</sub>* is a request to get or set information associated with the object. The following requests (defined in **UI\_WIN.HPP**) are recognized by the relative constraint 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 **RLCF\_FLAGS** that contains the flags to be cleared. 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 **RLCF\_FLAGS**. If *data* is NULL, a pointer to *rlcFlags* will be returned.

**I\_GET\_DENOMINATOR**—Returns the *denominator* value. If this message is sent, *data* must be a pointer to a variable of type **int** where the constraint's *denominator* will be copied.

**I\_GET\_NUMERATOR**—Returns the *numerator* value. If this message is sent, *data* must be a pointer to a variable of type **int** where the constraint's *numerator* will be copied.

**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 **RLCF\_FLAGS** that contains the flags to be set. This request only sets those flags that are passed in; it does not clear any flags that are already set.

**I\_SET\_DENOMINATOR**—Sets the *denominator* value. If this message is sent, *data* must be a pointer to a variable of type **int** that contains the constraint's new *denominator*.

**I\_SET\_NUMERATOR**—Sets the *numerator* size allowed by the constraint. If this message is sent, *data* must be a pointer to a variable of type **int** that contains the constraint's new *numerator*.

All other requests are passed to **UI\_CONSTRAINT::Information()** for processing.

- *data<sub>in/out</sub>* 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<sub>in</sub>* is not used.

## UI\_RELATIVE\_CONSTRAINT::Modify

### Syntax

```
#include <ui_win.hpp>

virtual void Modify(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 updates the managed object's position and, if necessary, size according to the constraints specified. The geometry manager calls each constraint's **Modify()** function whenever the parent object's position or size is changed.

## Storage Members

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

## UI\_RELATIVE\_CONSTRAINT::UI\_RELATIVE\_CONSTRAINT

### Syntax

```
#include <ui_win.hpp>

UI_RELATIVE_CONSTRAINT(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 `UI_RELATIVE_CONSTRAINT` by loading the object from a data file. Typically, the programmer does not need to use this constructor. If a constraint is stored in a data file it is usually stored as part of a geometry manager and will be loaded when the geometry manager is loaded.

- *name<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* 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`."
- *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."

- *objectTable<sub>in</sub>* 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 this manual. 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<sub>in</sub>* 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 this manual. 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.

## UI\_RELATIVE\_CONSTRAINT::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 UI\_RELATIVE\_CONSTRAINT from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* 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."
- *object<sub>in</sub>* 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."
- *objectTable<sub>in</sub>* 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 this manual. 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<sub>in</sub>* 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 *UIWINDOWOBJECT:-userTable* in "Chapter 43—UI\_WINDOW\_OBJECT" in this manual. 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.

## UI\_RELATIVE\_CONSTRAINT::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<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* 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`."
- *object<sub>in</sub>* 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`."
- *objectTable<sub>in</sub>* 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 `UIJWINDOWjOBJECT::objectTable` in "Chapter 43—`UI_WINDOW_OBJECT`" in this manual. 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<sub>in</sub>* 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 this manual. 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.

## UI\_RELATIVE\_CONSTRAINT::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<sub>out</sub>* is a pointer to the object's **New()** function.

## UI\_RELATIVE\_CONSTRAINT::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<sub>in</sub>* is the name of the object to be stored.
- *file<sub>in</sub>* 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."
- *object<sub>in</sub>* 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."
- *objectTable<sub>in</sub>* 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 this manual. 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<sub>in</sub>* 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 this manual. 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 39 - UI\_SCROLL\_INFORMATION

The UI\_SCROLL\_INFORMATION structure is used to maintain scroll information. It is used to send scrolling events to objects as well as to maintain current scroll settings.

The UI\_SCROLL\_INFORMATION structure is declared in **UI\_EVT.HPP**. Its public and protected members are:

```
struct ZIL_EXPORT_CLASS UI_SCROLL_INFORMATION
{
    ZIL_INT16 current;
    ZIL_INT16 minimum;
    ZIL_INT16 maximum;
    ZIL_INT16 showing;
    ZIL_INT16 delta;
};
```

### General Members

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

- *current* indicates the current scroll position within the range designated by *minimum* and *maximum*. If the values are settings for a UIW\_SCROLL\_BAR, *current* affects the relative position of the thumb button between the two end buttons of the scroll bar.
- *minimum* is the minimum value of the scroll range. The actual value used is insignificant, *minimum* is used in relation to *maximum* and *current*, so their relative values must make sense. The object using UI\_SCROLL\_INFORMATION is responsible for setting the values and interpreting their meaning.
- *maximum* is the maximum value of the scroll range. The actual value used is insignificant, *maximum* is used in relation to *minimum* and *current*, so their relative values must make sense. The object using UI\_SCROLL\_INFORMATION is responsible for setting the values and interpreting their meaning.
- *showing* indicates how much of the scroll range is "visible." *showing* controls how far *current* is moved when a full page scroll is performed. For example, if a text object has 100 lines of text, and 10 lines are visible, the scroll bar that controls it might have a range of 1 to 90 and a *showing* value of 10 (if 10 lines are visible then only 90 lines need to be scrolled for the entire field to be viewed). If the end-user selects a full page scroll on the scroll bar the text object will scroll by 10 lines. The

scroll bar's thumb button will also be adjusted, as will its *current* value, by 1/9th of the range (i.e., it moves by 10, the value of *showing*, within the range of 90 values).

- *delta* indicates how far to adjust the *current* value when the smallest scroll movement is made. For example, on the text object in the example above, *delta* will be 1, indicating that selecting an end button on the scroll bar will scroll the text 1 line.

## Portability

This structure is available on the following environments:

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

## Example

```
#include <ui_win.hpp>

EVENT_TYPE SCROLL_OBJECT::Event(const UI_EVENT &event)
{
    UI_WINDOW_OBJECT *object;

    // Switch on the event type.
    switch (ccode)
    {
    case S_HSCROLL:
    case S_VSCROLL:
        object = Current();
        int hDelta = 0, vDelta = 0;

        if (ccode == S_HSCROLL && hScroll)
        {
            hScroll->Event(event);
            hDelta = -event.scroll.delta * (object->trueRegion.right -
                object->trueRegion.left + 1);
        }
        else if (ccode == S_VSCROLL && vScroll)
        {
            vScroll->Event(event);
            vDelta = -event.scroll.delta * (object->trueRegion.bottom -
                object->trueRegion.top + 1);
        }

        break;
    }
}
```

```
    // Return the control code,  
    return (ccode);  
}
```



## CHAPTER 40 - UI\_TEXT\_DISPLAY

The UI\_TEXT\_DISPLAY class implements a text display that writes directly to screen memory. The UI\_TEXT\_DISPLAY is used for both DOS text mode applications and for Curses applications. Since the UI\_TEXT\_DISPLAY class is derived from the display class UI\_DISPLAY, only details specific to the UI\_TEXT\_DISPLAY class are given in this chapter. For descriptions and examples regarding virtual or inherited display members, see "Chapter 7—UI\_DISPLAY."

The UI\_TEXT\_DISPLAY class is declared in **UI\_DSP.HPP**. Its public and protected members are:

```
class ZIL_EXPORT CLASS UI_TEXT_DISPLAY : public UI_DISPLAY,
    public UI_REGION_LIST
{
public:
    TDM_MODE mode;

    UI_TEXT_DISPLAY(TDM_MODE mode = TDM_AUTO);
    virtual ~UI_TEXT_DISPLAY(void);
    virtual void Line(ZIL_SCREENID screenID, int column1, int line1,
        int column2, int line2, const UI_PALETTE *palette, int width = 1,
        int _xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual ZIL_COLOR MapColor(const UI_PALETTE *palette,
        int isForeground = 1);
    virtual void Rectangle(ZIL_SCREENID screenID, int left, int top,
        int right, int bottom, const UI_PALETTE *palette, int width = 1,
        int fill = FALSE, int _xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void RectangleXORDiff(const UI_REGION &oldRegion,
        const UI_REGION &newRegion, ZIL_SCREENID screenID = ID_SCREEN,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void RegionDefine(ZIL_SCREENID screenID, int left, int top,
        int right, int bottom);
    virtual void RegionMove(const UI_REGION &oldRegion, int newColumn,
        int newLine, ZIL_SCREENID oldScreenID = ID_SCREEN,
        ZIL_SCREENID newScreenID = ID_SCREEN);
    virtual void Text(ZIL_SCREENID screenID, int left, int top,
        const ZIL_ICHAR *text, const UI_PALETTE *palette, int length = -1,
        int fill = TRUE, int _xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
        ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
    virtual int TextHeight(const ZIL_ICHAR *string,
        ZIL_SCREENID screenID = ID_SCREEN,
        ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
    virtual int TextWidth(const ZIL_ICHAR *string,
        ZIL_SCREENID screenID = ID_SCREEN,
        ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
    virtual int VirtualGet(ZIL_SCREENID screenID, int left, int top,
        int right, int bottom);
    virtual int VirtualPut(ZIL_SCREENID screenID);

protected:
    ZIL_SCREEN_CELL *_screen;
    int _virtualCount;
    UI_REGION _virtualRegion;
    char _stopDevice;
```

```

    // I18N member variables and functions,
public:
    static ZIL_ICHAR _tCornerUL[];
    static ZIL_ICHAR _tCornerUR[];
    static ZIL_ICHAR _tCornerLL[];
    static ZIL_ICHAR _tCornerLR[];
    static ZIL_ICHAR _tHorizontal[];
    static ZIL_ICHAR _tVertical[];
};

```

## General Members

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

- *mode* is the text mode that is initialized.
- *\_screen* is a pointer to the BIOS screen buffer.
- *\_moveBuffer* is a pointer to screen memory. This extra memory facilitates faster screen moves.
- *\_virtualCount* is a count of the number of virtual screen operations that have taken place. For example, when the **VirtualGet()** function is called, *\_virtualCount* is decremented. Additionally, when the **VirtualPut()** function is called, *\_virtualCount* is incremented.
- *\_virtualRegion* is the region affected by either **VirtualGet()** or **VirtualPut()**.
- *\_stopDevice* is a variable used to prevent recursive updates of device images on the display. If *\_stopDevice* is TRUE, no drawing will be done to the screen. Otherwise, drawing will be made directly to the screen display.

## UI\_TEXT\_DISPLAY::UI\_TEXT\_DISPLAY

### Syntax

```
#include <ui_dsp.hpp>
```

```
UI_TEXT_DISPLAY(TDM_MODE mode = TDM_AUTO);
```

## 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 `UI_TEXT_DISPLAY` object. When a new `UI_TEXT_DISPLAY` class is constructed, the system clears the screen to the background color and pattern specified by the inherited palette variable `backgroundPalette`. See "Chapter 27—UI\_PALETTE" of this manual for more information about palettes. Also, the blink attribute is disabled to allow the use of high-intensity colors.

- `_modein` specifies the type of text display to create. The available display modes (defined in `UI_DSP.HPP`) are:

**TDM\_AUTO**—Creates a text display using the current text mode.

**TDM\_25x40** and **TDM\_BW\_25x40**—Create a text display with 25 lines and 40 columns.

**TDM\_25x80**, **TDM\_BW\_25x80** and **TDM\_MONO\_25x80**—Create a text display with 25 lines and 80 columns.

**TDM\_43x80**—Creates a text display with 43 lines and 80 columns on an EGA display or 50 lines and 80 columns on a VGA display.

## Example

```
#include <ui_win.hpp>

main()
{
    // Initialize OpenZincApplicationFramework.
    UI_DISPLAY *display = new UI_GRAPHICS_DISPLAY;
    if (!display->installed)
    {
        delete display;
        display = new UI_TEXT_DISPLAY;
    }

    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
    + new UID_KEYBOARD
    + new UID_MOUSE
    + new UID_CURSOR;
}
```



```

UI_WINDOW_MANAGER *windowManager =
    new UI_WINDOW_MANAGER(display, eventManager);

// Restore the system,
delete windowManager;
delete eventManager;
delete display;
return (0);
}

```

## UI\_TEXT\_DISPLAY::~~UI\_TEXT\_DISPLAY

### Syntax

```

#include <ui_dsp.hpp>

~UI_TEXT_DISPLAY(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 UI\_TEXT\_DISPLAY class. Care should be taken to only destroy a UI\_TEXT\_DISPLAY class that is not attached to another associated object.

## Internationalization Members

This section describes those members used for internationalization purposes.

- `_tCornerUL` is an array that contains the characters used to represent the upper left corner of a window in text mode. By default these characters are 'r' and 'ff'.
- `_tCornerUR` is an array that contains the characters used to represent the upper right corner of a window in text mode. By default these characters are ' ' and '\n'.

- *\_tCornerLL* is an array that contains the characters used to represent the lower left corner of a window in text mode. By default these characters are ‘└’ and ‘┘’.
- *\_tCornerLR* is an array that contains the characters used to represent the lower right corner of a window in text mode. By default these characters are ‘┐’ and ‘┑’.
- *\_tHorizontal* is an array that contains the characters used to represent the horizontal (i.e., top and bottom) edges of a window in text mode. By default these characters are ‘—’ and ‘=’.
- *\_tVertical* is an array that contains the characters used to represent the vertical (i.e., left and right) edges of a window in text mode. By default these characters are ‘|’ and ‘||’.



## CHAPTER 41 - UI\_WCC\_DISPLAY

The UI\_WCC\_DISPLAY class object is a graphics display class that uses the graphics library packaged with the Watcom compiler. Since the UI\_WCC\_DISPLAY class is derived from UI\_DISPLAY, only details specific to the UI\_WCC\_DISPLAY class are given in this chapter. For descriptions and examples regarding virtual or inherited display members, see "Chapter 7—UI\_DISPLAY."

The UI\_WCC\_DISPLAY class is declared in **UI\_DSP.HPP**. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UI_WCC_DISPLAY : public UI_DISPLAY,
    public UI_REGION_LIST
{
public:
    Struct ZIL_EXPORT_CLASS WCCFONT
    {
        char *typeFace;
        char *options;
    };
    typedef unsigned char WCCPATTERN[8];

    static UI_PATH *searchPath;
    static WCCFONT fontTable[ZIL_MAXFONTS];
    static WCCPATTERN patternTable[ZIL_MAXPATTERNS];

    UI_WCC_DISPLAY(int mode = 0);
    virtual ~UI_WCC_DISPLAY(void);
    virtual void Bitmap(ZIL_SCREENID screenID, int column, int line,
        int bitmapWidth, int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette = ZIL_NULLP(UI_PALETTE),
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
        ZIL_BITMAP_HANDLE *colorBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE),
        ZIL_BITMAP_HANDLE *monoBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE));
    virtual void BitmapArrayToHandle(ZIL_SCREENID screenID, int bitmapWidth,
        int bitmapHeight, const ZIL_UINT8 *bitmapArray,
        const UI_PALETTE *palette, ZIL_BITMAP_HANDLE *colorBitmap,
        ZIL_BITMAP_HANDLE *monoBitmap);
    virtual void BitmapHandleToArray(ZIL_SCREENID screenID,
        ZIL_BITMAP_HANDLE colorBitmap, ZIL_BITMAP_HANDLE monoBitmap,
        int *bitmapWidth, int *bitmapHeight, ZIL_UINT8 **bitmapArray);
    virtual void Ellipse(ZIL_SCREENID screenID, int column, int line,
        int startAngle, int endAngle, int xRadius, int yRadius,
        const UI_PALETTE *palette, int fill = FALSE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual void IconArrayToHandle(ZIL_SCREENID screenID, int iconWidth,
        int iconHeight, const ZIL_UINT8 *iconArray,
        const UI_PALETTE *palette, ZIL_ICON_HANDLE *icon);
    virtual void IconHandleToArray(ZIL_SCREENID screenID,
        ZIL_ICON_HANDLE icon, int *iconWidth, int *iconHeight,
        ZIL_UINT8 **iconArray);
    virtual void Line(ZIL_SCREENID screenID, int column1, int line1,
        int column2, int line2, const UI_PALETTE *palette, int width = 1,
        int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
    virtual ZIL_COLOR MapColor(const UI_PALETTE *palette, int isForeground);
    virtual void Polygon(ZIL_SCREENID screenID, int numPoints,
        const int *polygonPoints, const UI_PALETTE *palette,
        int fill = FALSE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
```

```

virtual void Rectangle(ZIL_SCREENID screenID, int left, int top,
    int right, int bottom, const UI_PALETTE *palette, int width = 1,
    int fill = FALSE, int xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void RectangleXORDiff(const UI_REGION &oldRegion,
    const UI_REGION knewRegion, ZIL_SCREENID screenID = ID_SCREEN,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void RegionDefine(ZIL_SCREENID screenID, int left, int top,
    int right, int bottom);
virtual void RegionMove(const UI_REGION &oldRegion, int newColumn,
    int newLine, ZIL_SCREENID oldScreenID = ID_SCREEN,
    ZIL_SCREENID newScreenID = ID_SCREEN);
virtual void Text(ZIL_SCREENID screenID, int left, int top,
    const ZIL_ICHAR *text, const UI_PALETTE *palette, int length = -1,
    int fill = TRUE, int xor = FALSE,
    const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextHeight(const ZIL_ICHAR *string,
    ZIL_SCREENID screenID = ID_SCREEN,
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextWidth(const ZIL_ICHAR *string,
    ZIL_SCREENID screenID = ID_SCREEN,
    ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int VirtualGet(ZIL_SCREENID screenID, int left, int top,
    int right, int bottom);
virtual int VirtualPut(ZIL_SCREENID screenID);

protected:
    int maxColors;
    signed char virtualCount;
    UI_REGION virtualRegion;
    char stopDevice;
    int fillPattern;
    int backgroundColor;
    int foregroundColor;

    void SetFont(ZIL_LOGICAL_FONT logicalFont);
    void SetPattern(const UI_PALETTE *palette, int xor);
};

```

## General Members

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

- *WCCFONT* is a structure that contains the following font information:

*typeFace* contains the string name of the font. OpenZinc uses Watcom's Helvetica font, so for the three fonts defined by OpenZinc, *typeFace* is "Helv."

*options* contains the font characteristics. For more information see **`_setfont()`** in the *Watcom C Graphics Library Reference*.

- *WCCPATTERN* is an array of 8 bytes that make up the 8x8 bitmap pattern. Each byte (8 bits) corresponds to 8 pixels in the pattern. The patterns defined by OpenZinc are: PTN\_SOLID\_FILL, PTN\_INTERLEAVE\_FILL and PTN\_BACKGROUND\_FILL.

For more information see **setfillpattern()** in the *Watcom C Graphics Library Reference*.

- *searchPath* contains the path to be searched for the font file. The WCC graphics library needs to access font files at run-time so that it can draw characters in graphics mode. Because OpenZinc uses Watcom's Helvetica font, the `UI_WCC_DISPLAY` class needs to find the `HELYB.FON` file at run-time. If the display class cannot find this file, graphics mode will not initialize.
- *fontTable* is an array of `WCCFONT`. The default array contains space for 10 `WCCFONT` entries. The following entries are pre-defined by OpenZinc:

`FNT_SMALL_FONT`—A font that is used to display an icon's text string.

`FNT_DIALOG_FONT`—A font that is used when text is displayed on window objects (e.g., `UIW_BUTTON`, `UIW_STRING`, `UIW_TEXT`, etc.)

`FNT_SYSTEM_FONT`—A sans-serif style font that is used to display a window's title.

NOTE: To use these fonts, or if other "stroked" fonts are added to this table, the proper Watcom font files must be in the current working directory or in the environment's path at run-time.

See the description of the `UI_WINDOW_OBJECT::font` member variable in "Chapter 43—`UI_WINDOW_OBJECT`" for information on specifying which font an object uses.

- *patternTable* is an array of `WCCPATTERN`. The default array contains space for 15 `WCCPATTERN` entries. The following entries are pre-defined by OpenZinc:

`PTN_SOLID_FILL`—Solid fill.

`PTN_INTERLEAVE_FILL`—Interleaving line fill.

`PTN_BACKGROUND_FILL`—Background fill style.

- *maxColors* is the maximum number of colors supported by the graphics mode that was initialized. For example, an EGA display might support sixteen colors. This member will be filled in according to information obtained from the WCC graphics library after it has initialized. The WCC graphics library supports SVGA modes,

including 256 color mode. OpenZinc will support whatever mode is initialized by the WCC graphics library.

- `_virtualCount` is a count of the number of virtual screen operations that have taken place. For example, when the `VirtualGet()` function is called, `_virtualCount` is decremented. Additionally, when the `VirtualPut()` function is called, `_virtualCount` is incremented.
- `_virtualRegion` is the region affected by either `VirtualGet()` or `VirtualPut()`.
- `_stopDevice` is a variable used to prevent recursive updates of device images on the display. If `_stopDevice` is TRUE, no drawing will be done to the screen. Otherwise, drawing will be made directly to the screen display.
- `_fillPattern` is an index into the `patternTable` specifying the current fill pattern.
- `_backgroundColor` is the current background drawing color.
- `_foregroundColor` is the current foreground drawing color.

## UI\_WCC\_DISPLAY::UI\_WCC\_DISPLAY

### Syntax

```
#include <ui_dsp.hpp>

UI_WCC_DISPLAY(int mode = 0);
```

### 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 `UI_WCC_DISPLAY` object. When a new `UI_WCC_DISPLAY` class is constructed, the screen display is set to the background color and pattern specified by the inherited variable `backgroundPalette`.

- *mode<sub>m</sub>* specifies the graphics mode that should be initialized. If *mode* is 0, which is the default, the `UI_WCC_DISPLAY` class will initialize the highest resolution graphics mode possible using the `WCC_MAXRESMODE` constant. For more information on the possible values for *mode*, see `_setvideomode()` in the *Watcom C Graphics Library Reference*.

## Example

```
#include <ui_win.hpp>

main()
{
    // Initialize OpenZinc Application Framework.
    UI_DISPLAY *display = new UI_WCC_DISPLAY;

    // Restore the system,
    delete windowManager;
    delete eventManager;
    delete display;
    return (0);
}
```

## UI\_WCC\_DISPLAY::~~UI\_WCC\_DISPLAY

### Syntax

```
#include <ui_dsp.hpp>

~UI_WCC_DISPLAY(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 `UI_WCC_DISPLAY` class. Care should be taken to only destroy a `UI_WCC_DISPLAY` class that is not attached to another associated object.



## UI\_WCC\_DISPLAY::SetFont

### Syntax

```
#include <ui_dsp.hpp>

void SetFont(ZIL_LOGICAL_FONT logicalFont);
```

### 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 font information used by the WCC graphics library. The information contained in the *logicalFont* entry of the *fontTable* array is used to set the font.

- *logicalFont<sub>in</sub>* is the font to be used. *logicalFont* is an entry into the *fontTable* array.

## UI\_WCC\_DISPLAY::SetPattern

### Syntax

```
#include <ui_dsp.hpp>

void SetPattern(const UIJPALETTE *palette, int _xor);
```

### Portability

This function is available on the following environments:

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

## Remarks

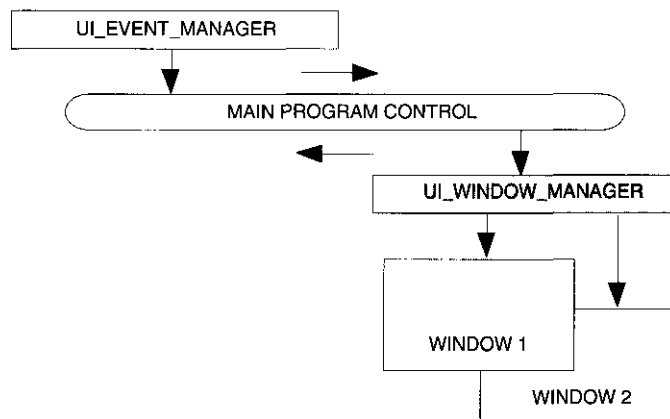
This function is used to set the pattern information used by the WCC graphics library. The information contained in *palette* is used to set the pattern.

- *palette* contains the pattern style, foreground color, and background color to be used when setting the pattern.
- *\_xor\_in* indicates if the pattern should be drawn with the xor attribute on. If *\_xor* is TRUE, the pattern will be an xor pattern. Otherwise, the pattern will not be xor.



## CHAPTER 42 - UI\_WINDOW\_MANAGER

The UI\_WINDOW\_MANAGER class is used to maintain the list of windows displayed by the application and to dispatch events. The Window Manager also handles some events that are generic to the entire application. For example, the user may select an option that causes the application to close. The Window Manager can detect this event and begin the process to close the application. The graphic illustration below shows the conceptual operation of the Window Manager within the library:



The UI\_WINDOW\_MANAGER class is declared in UI\_WIN.HPP. Its public and protected members are:

```
Class ZIL_EXPORT_CLASS UI_WINDOW_MANAGER : public UIW_WINDOW
{
public:
    ZIL_EXIT_FUNCTION exitFunction;
    UI_WINDOW_OBJECT *dragObject;

    UI_WINDOW_MANAGER(UI_DISPLAY *display, UI_EVENT_MANAGER *eventManager,
        ZIL_EXIT_FUNCTION exitFunction = ZIL_NULLF(ZIL_EXIT_FUNCTION));
    virtual ~UI_WINDOW_MANAGER(void);
    void Center(UI_WINDOW_OBJECT *object);
    virtual EVENT_TYPE Event(const UI_EVENT &event);
    virtual void Information(ZIL_INFO_REQUEST request, void *data,
        ZIL_OBJECTID objectID = ID_DEFAULT);

    // List members.
    UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);
    UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT *object);
    UI_WINDOW_MANAGER &operator+(UI_WINDOW_OBJECT *object);
    UI_WINDOW_MANAGER koperator-(UI_WINDOW_OBJECT *object);
};
```

## General Members

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

- *exitFunction* is a programmer defined function that is called whenever the Window Manager receives the L\_EXIT\_FUNCTION message. For example, the programmer may want to confirm whether the end-user really wants to exit the program. The programmer can use the *exitFunction* to add a window to the Window Manager confirming the end-user's desire to exit. If *exitFunction* is NULL, the L\_EXIT\_FUNCTION message is changed to an L\_EXIT message by the Window Manager. The definition of the *exitFunction* is as follows:

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

*returnValue<sub>out</sub>* indicates what the program should do. *returnValue* is returned to the main event loop, so if the program should terminate immediately, the function should return an L\_EXIT, which will cause the main event loop to exit and the program to end. If some other action is desired, the function may place one or more events on the queue. In this case, *returnValue* should be S\_CONTINUE or something similar.

*display<sub>in</sub>* is a pointer to the display.

*eventManager<sub>in</sub>* is a pointer to the Event Manager.

*windowManager<sub>in</sub>* is a pointer to the Window Manager.

It is also possible to have the *exitFunction* called when a particular window is closed. To accomplish this, the Window Manager's *screenID* must be set equal to the window's *screenID*. The following piece of code demonstrates this:

```
*windowManager  
+ window;  
  
windowManager->screenID = window->screenID;
```

- *dragObject* is a pointer to the object that is being dragged if a drag and drop operation is in progress.

## UI\_WINDOW\_MANAGER::UI\_WINDOW\_MANAGER

### Syntax

```
#include <ui_win.hpp>

UI_WINDOW_MANAGER(UI_DISPLAY *display,
                  UI_EVENT_MANAGER *eventManager,

                  ZIL_EXIT_FUNCTION exitFunction = ZIL_NULLF(ZIL_EXIT_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 UI\_WINDOW\_MANAGER object. It should be called after the display and Event Manager classes have been called.

- *display<sub>in</sub>* is a pointer to the display. This pointer is used by window objects when they draw.
- *eventManager<sub>in</sub>* is a pointer to the Event Manager. This pointer is used by window objects to place events on the queue or to send messages to devices.
- *exitFunction<sub>in</sub>* is a programmer defined function that is called whenever the Window Manager receives the L\_EXIT\_FUNCTION message. See the description of the *exitFunction* member above for more details.

### Example

```
#include <ui_win.hpp>

main()
{
    // Initialize the system.
    UI_DISPLAY *display = new UI_TEXT_DISPLAY();
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    UI_WINDOW_MANAGER *windowManager = new UI_WINDOW_MANAGER(display,
        eventManager);
}
```

```

    // Restore the system,
    delete windowManager;
    delete eventManager;
    delete display;
    return (0);
}

```

## **UI\_WINDOW\_MANAGER::~~UI\_WINDOW\_MANAGER**

### **Syntax**

```

#include <ui_win.hpp>

virtual ~UI_WINDOW_MANAGER(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 UI\_WINDOW\_MANAGER object. Destroying the Window Manager will also delete all windows still attached to the Window Manager unless they have the WOAF\_NO\_DESTROY flag set. If this flag is set on a window, the programmer is responsible for deleting the window.

### **Example**

```

#include <ui_win.hpp>

main()
{
    // Initialize the system.
    UI_DISPLAY *display = new UI_TEXT_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    UI_WINDOW_MANAGER *windowManager = new UI_WINDOW_MANAGER(display,
        eventManager);

    // Restore the system,
    delete windowManager;
}

```

```

delete eventManager;
delete display;
return (0);
}

```

## **UI\_WINDOW\_MANAGER::Add** **UI\_WINDOW\_MANAGER: operator +**

### **Syntax**

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

```
UI_WINDOW_OBJECT *Add(UI_WINDOW_OBJECT *object);
or
```

```
UI_WINDOW_MANAGER &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 window to the UI\_WINDOW\_MANAGER object.

The first overloaded function adds a window to the UI\_WINDOW\_MANAGER object. The window will be the current window. This function can be used to add a new window to the Window Manager or to make an already displayed window current.

- *returnValue<sub>out</sub>* is a pointer to *object* if the addition was successful. Otherwise, *returnValue* is NULL.
- *object<sub>in</sub>* is a pointer to the window to be added to the Window Manager.

The second operator overload adds a window to the UI\_WINDOW\_MANAGER object. This operator overload is equivalent to calling the **UI\_WINDOW\_MANAGER::Add( )** function except that it allows the chaining of window additions to the UI\_WINDOW\_MANAGER object.



- *returnValue<sub>out</sub>* is a pointer to the UI\_WINDOW\_MANAGER object. This pointer is returned so that the operator may be used in a statement containing other operations.
- *object<sub>in</sub>* is a pointer to the new window that is to be added to the Window Manager.

## 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;
}
```

## UI\_WINDOW\_MANAGER::Center

### Syntax

```
#include <ui_win.hpp>

void Center(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 centers an object on the screen. Only objects that are attached directly to the Window Manager should be centered using this function (i.e., a button attached to a window cannot be centered using this function).

- *object<sub>in</sub>* is a pointer to the object that is to be centered on the screen, *object* must be attached directly to the Window Manager.

## UI\_WINDOW\_MANAGER::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 events sent to the Window Manager and dispatches events destined for other objects. In the main event loop, events that are waiting to be processed are removed from the Event Manager's event queue and sent to this function. If the event is meant for the Window Manager, it is processed by this function. If the event is intended for another object the **Event()** function dispatches it. How the **Event()** function dispatches events depends on the event and the environment. If the event is a OpenZinc event—meaning it was not generated by the operating system—the event is routed to the appropriate window. The event will be processed in a top-down fashion, with the top-level objects (e.g., a window attached to the Window Manager) getting the event before the sub-objects (e.g., a button attached to a window) will. If the environment is a graphical operating system that already has an event-driven messaging system, such as Windows or Motif, and the event came from the operating system, the event will be passed to the operating system so that it may dispatch it as it normally would. Thus, it is possible to place events on the queue that are specific to your application but still interact with the operating system's API if desired.

- *returnValue<sub>out</sub>* indicates how *event* was processed. If the event is processed successfully, the function returns the logical type of event that was interpreted from *event*. The event may have been processed by the **UI\_WINDOW\_MANAGER::Event()** function directly or it may have been dispatched by the Window Manager and handled by another object's **Event()** function. See "Appendix B—System Events" of *Programmer's Reference Volume 2* for a complete listing of system events and "Appendix C—Logical Events" of *Programmer's Reference Volume 2* for a complete listing of logical events. The following event types (declared in

**UI\_EVT.HPP**) may need to be handled specially when returned from this function to the main event loop:

**L\_EXIT**—The Window Manager received an event that either mapped to the L\_EXIT event, or an action was performed that caused the Window Manager to generate the L\_EXIT event. If this event is returned, program execution should be discontinued.

**S\_NO\_OBJECT**—There are no objects in the Window Manager's list. This message is returned whenever the message is object-specific but no object is attached to the Window Manager. Typically, the application should end if this message is received.

**S\_UNKNOWN**—The event could not be processed. This may be the result of an invalid operation or because the event was not recognized by any objects.

- *event<sub>in</sub>* contains a run-time message. The type of operation performed depends on the interpretation of the event. The following logical events are processed by **Event()**:

**L\_BEGIN\_COPY\_DRAG**—Indicates that the end-user is beginning a drag operation to copy an object.

**L\_BEGIN\_MOVE\_DRAG**—Indicates that the end-user is beginning a drag operation to move an object.

**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\_COPY\_DRAG**—Indicates that the end-user is continuing a drag operation to copy an object.

**L\_CONTINUE\_MOVE\_DRAG**—Indicates that the end-user is continuing a drag operation to move an object.

**L\_END\_COPY\_DRAG**—Indicates that the end-user has completed a drag operation to copy an object.

**L\_END\_MOVE\_DRAG**—Indicates that the end-user has completed a drag operation to move an object.

**L\_EXIT**—Indicates that the end-user performed some action that should result in the termination of the program. The Window Manager does not actually process this event, but instead returns it to the main event loop which should process it. This event may be placed directly on the event queue by the programmer or it may be interpreted from an event that resulted from the end-user's actions.

**L\_EXIT\_FUNCTION**—Causes the Window Manager to call the programmer-defined exit function, if one exists. The most common use for the exit function is to confirm that the user wants to exit the program. If no exit function was provided by the programmer this message will result in an L\_EXIT being processed. See the description of the *exitFunction* member above for more details.

**L\_HELP**—If the message is interpreted by the Window Manager, it requests general help associated with the application. If this message is interpreted by a particular window object, it requests the context-sensitive help associated with the object.

**L\_MAXIMIZE**—Maximizes the current window. Typically, this event is the result of the end-user selecting a key combination to maximize the current window. This event may be placed directly on the event queue by the programmer or it may be interpreted from an event that resulted from the end-user's actions.

**L\_MINIMIZE**—Minimizes the current window. Typically, this event is the result of the end-user selecting a key combination to minimize the current window. This event may be placed directly on the event queue by the programmer or it may be interpreted from an event that resulted from the end-user's actions.

**L\_MOVE**—Allows the end-user to move the window from keyboard control. Typically, this event is the result of the end-user selecting a key combination to move the current window. This event may be placed directly on the event queue by the programmer or it may be interpreted from an event that resulted from the end-user's actions.

**L\_NEXT\_WINDOW**—Causes the next window to be made current. This event may be placed directly on the event queue by the programmer or it may be interpreted from an event that resulted from the end-user's actions.

**L\_RESTORE**—Restores the current window from its minimized or maximized state. Typically, this event is the result of the end-user selecting a key combination to restore the current window. This event may be placed directly on the event queue by the programmer or it may be interpreted from an event that resulted from the end-user's actions.

**L\_SIZE**—Allows the end-user to size the window from keyboard control. Typically, this event is the result of the end-user selecting a key combination to size the current window. This event may be placed directly on the event queue by the programmer or it may be interpreted from an event that resulted from the end-user's actions.

**L\_VIEW**—A general event that indicates the mouse was moved while no buttons were pressed. An object can use this message to change the appearance of the mouse pointer. This event is interpreted from an event generated by the mouse device.

**S\_ADD\_OBJECT**—Is used to add an object to the Window Manager. A pointer to the object to be added 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\_CASCADE**—Causes the Window Manager to size and arrange the windows in a cascaded fashion.

**S\_CLOSE**—Causes the Window Manager to close the current window, if possible. The current window will not be closed if the window has the `WOAF_LOCKED` flag set. In addition to closing the window, the Window Manager will also delete the window, freeing the memory. It will not delete the window, however, if the window has the `WOAF_NO_DESTROY` flag set. In this case the programmer is responsible for deleting the window. If there are any temporary windows attached to the Window Manager (i.e., windows that have the `WOAF_TEMPORARY` flag set) they will also be closed and deleted in addition to the first non-temporary window. This event may be placed directly on the event queue by the programmer or it may be interpreted from an event that resulted from the end-user's actions. This event is not sent to the window. The window will receive an `S_DEINITIALIZE` message when it is being closed.

**S\_CLOSE\_TEMPORARY**—Closes the current window if it is temporary. For example, a `UIW_POP_UP_MENU` is a temporary window. A temporary window is a window that has the `WOAF_TEMPORARY` flag set. If the current window is not a temporary window, no action will occur. This event may be

placed directly on the event queue by the programmer or it may be interpreted from an event that resulted from the end-user's actions.

**S\_REDISPLAY**—Causes a refresh of the display. All windows attached to the Window Manager will be redrawn. In some operating systems (e.g., DOS, Curses) the background may be redrawn as well.

**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 an object from the Window Manager. A pointer to the object to be subtracted 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.).

**NOTE:** Because most graphical operating systems already process their own events related to this object, or because some of the events listed above may not make sense for some of these operating systems, the messages listed above may not be handled in every environment. Wherever possible, Zinc allows the operating system to process its own messages so that memory use and speed will be as efficient as possible.

## Example

```
#include <ui_win.hpp>

main()
{

    // Get events until the L_EXIT event occurs.
    EVENT_TYPE ccode;
    do
    {
        UI_EVENT event;
        eventManager->Get(event, Q_NORMAL);
        ccode = windowManager->Event(event);
    } while (ccode != L_EXIT && ccode != S_NO_OBJECT);

}
```

## UI\_WINDOW\_MANAGER::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<sub>out</sub>* 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<sub>in</sub>* 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\_COPY\_TEXT**—Copies the text associated with the object. If this request is sent, *data* should be a pointer to a buffer of **ZIL\_ICHAR**. The text is used to identify the task in the Program Manager's task list. This request is specific to MS Windows.

**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 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\_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. The text is used to identify the task in the Program Manager's task list. This request is specific to MS Windows.

**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. The text is used to identify the task in the Program Manager's task list. This request is specific to MS Windows.

- *data<sub>in/out</sub>* 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<sub>in</sub>* 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>

UI_WINDOW_OBJECT *GetObject(char *name, UI_WINDOW_MANAGER *windowManager)
{
    // Find the window object given a name and return a pointer to it.
    return (windowManager->Information(I_GET_STRINGID_OBJECT, name,
        ID_WINDOW_MANAGER));
}
```



## **UI\_WINDOW\_MANAGER::Subtract** **UI\_WINDOW\_MANAGER: operator -**

### **Syntax**

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

```
UI_WINDOW_OBJECT *Subtract(UI_WINDOW_OBJECT *object);  
or
```

```
UI_WINDOW_MANAGER &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 window from the UI\_WINDOW\_MANAGER object. They only remove the window from the Window Manager list—they do not delete the object.

The first function subtracts a window from the UI\_WINDOW\_MANAGER object. The window will not be deleted. The programmer is responsible for deletion of each object explicitly subtracted from a list.

- *returnValue<sub>out</sub>* is a pointer to the next window in the list. This value is NULL if there are no more windows after the subtracted window.
- *element<sub>in</sub>* is a pointer to the window to be subtracted from the list.

The second operator overload removes a window from the UI\_WINDOW\_MANAGER object. The window will not be deleted. The programmer is responsible for deletion of each object explicitly subtracted from a list. This operator overload is equivalent to calling the **Subtract^** ) function, except that it allows the chaining of list element removals from the UI\_WINDOW\_MANAGER object.

- *returnValue<sub>out</sub>* is a pointer to the UI\_WINDOW\_MANAGER object. This pointer is returned so that the operator may be used in a statement containing other operations.

- *object*^ is a pointer to the window that is to be removed from the list.

## Example

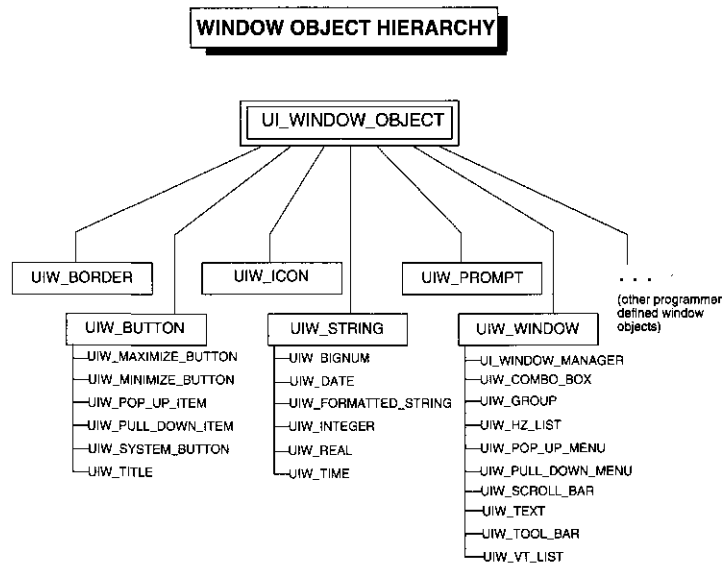
```
#include <ui_win.hpp>

ExampleFunction(UI_WINDOW_MANAGER *windowManager, UIW_WINDOW *window)
{
    #windowManager - window;
}
```



## CHAPTER 43 - UI\_WINDOW\_OBJECT

The UI\_WINDOW\_OBJECT class is the base class to all window objects. It provides the basic functionality required for objects to be displayed. It should not be used as a constructed class. Rather, derived classes, such as UIW\_BORDER, UIW\_BUTTON or UIW\_WINDOW must be used. The figure below shows the window object hierarchy:



Windows and window objects are attached to the Window Manager or a window at runtime by the programmer. Once a window or window object is attached, it receives event information from the Window Manager.

The UI\_WINDOW\_OBJECT class is declared in **UI\_WIN.HPP**. Its public and protected members are:

```

class ZIL_EXPORT_CLASS UI_WINDOW_OBJECT : public UI_ELEMENT,
    public ZIL_INTERNATIONAL
{
public:
    // Forward declaration of classes used by UI_WINDOW_OBJECT.
    friend class ZIL_EXPORT_CLASS UI_WINDOW_MANAGER;
    friend class ZIL_EXPORT_CLASS UI_ERROR_STUB;
    friend class ZIL_EXPORT_CLASS UI_ERROR_SYSTEM;
    friend class ZIL_EXPORT_CLASS UI_HELP_STUB;
    friend class ZIL_EXPORT_CLASS UI_HELP_SYSTEM;

    static int repeatRate;
    static int doubleClickRate;
    static WOS_STATUS defaultStatus;
  
```

```

static UI_DISPLAY *display;
static UI_EVENT_MANAGER *eventManager;
static UI_WINDOW_MANAGER *windowManager;
static UI_ERROR_STUB *errorSystem;
static UI_HELP_STUB *helpSystem;
static ZIL_STORAGE_READ_ONLY *defaultStorage;
static UI_ITEM *objectTable;
static UI_ITEM *userTable;
static ZIL_ICHAR _className[];

UI_EVENT_MAP *eventMapTable;
UI_EVENT_MAP *hotKeyMapTable;
UI_PALETTE_MAP *paletteMapTable;

#if defined(ZIL_MACINTOSH)
union
{
    ZIL_SCREENID screenID;
    ControlHandle controlScreenID;
    ListHandle listScreenID;
    MenuHandle menuScreenID;
    TEHandle textScreenID;
    WindowPtr windowScreenID;
};
#else
ZIL_SCREENID screenID;
#endif
WOF_FLAGS woFlags;
WOAF_FLAGS woAdvancedFlags;
#if defined(ZIL_EDIT)
WOAF_FLAGS designerAdvancedFlags;
#endif
WOS_STATUS woStatUS;
UI_REGION trueRegion;
UI_REGION relative;
UI_WINDOW_OBJECT *parent;
UI_HELP_CONTEXT helpContext;

UIF_FLAGS userFlags;
UIS_STATUS userStatus;
void *userObject;
EVENT_TYPE (*userFunction)(UI_WINDOW_OBJECT *object, UI_EVENT &event,
EVENT_TYPE ccode);
EVENT_TYPE UserFunction(const UI_EVENT &event, EVENT_TYPE ccode);

virtual ~UI_WINDOW_OBJECT(void);
virtual ZIL_ICHAR *ClassName(void);
virtual EVENT_TYPE Event(const UI_EVENT &event);
ZIL_LOGICAL_FONT Font(ZIL_LOGICAL_FONT font = FNT_NONE);
UI_WINDOW_OBJECT *Get(const ZIL_ICHAR *name);
UI_WINDOW_OBJECT *Get(ZIL_NUMBERID numberID);
unsigned HotKey(unsigned hotKey = 0);
unsigned HotKey(ZIL_ICHAR *text);
virtual void information(ZIL_INFO_REQUEST request, void *data,
ZIL_OBJECTID objectID = ID_DEFAULT);
int Inherited(ZIL_OBJECTID matchID);
EVENT_TYPE LogicalEvent(const UI_EVENT &event,
ZIL_OBJECTID currentID = 0, int nativeType = TRUE);
UI_PALETTE *LogicalPalette(LOGICAL_EVENT logicalEvent,
ZIL_OBJECTID currentID = 0);
NUMBERID NumberID(NUMBERID numberID = 0);
EVENT_TYPE RedisplayType(void);
void RegionConvert(UI_REGION ^region, int absolute);
UI_WINDOW_OBJECT *Root(int mdiChild = FALSE);
ZIL_OBJECTID SearchID(void);
ZIL_ICHAR *StringID(const ZIL_ICHAR *stringID = ZIL_NULLP(ZIL_ICHAR));
#if defined(ZIL_MOTIF)
static XmString CreateMotifString(ZIL_ICHAR *,

```

```

        ZIL_ICHAR ** = ZIL_NULLP(ZIL_ICHAR *), int = TRUE);
        static void StripHotKeyMark(ZIL_ICHAR *text);
        virtual ZIL_SCREENID TopWidget(void);
        Widget shell;
    #endif
    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));
    UI_WINDOW_OBJECT(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 = 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

    // List members.
    UI_WINDOW_OBJECT *Next(void);
    UI_WINDOW_OBJECT *Previous(void);

protected:
    ZIL_OBJECTID searchID;
    ZIL_NUMBERID numberID;
    ZIL_ICHAR stringID[32];
    ZIL_OBJECTID windowID[5];

    unsigned hotKey;
    ZIL_LOGICAL_FONT font;
    UI_PALETTE *lastPalette;
    ZIL_ICHAR *userObjectName; // Used for storage purposes only.
    ZIL_ICHAR *userFunctionName; // Used for storage purposes only.
    UI_REGION clip;
    #if defined(ZIL MSDOS) || defined(ZIL CURSES)
    static ZIL_ICHAR *pasteBuffer; // There is only one global paste buffer,
    static int pasteLength;
    #elif defined(ZIL WINNT)
    DWORD dwStyle;
    WNDPROC defaultCallback;
    void RegisterObject(char *className, char *winClassName,
        WNDPROC *defProcInstance, ZIL_ICHAR *title = ZIL_NULLP(ZIL_ICHAR),
        HMENU menu = 0);
    #elif defined(ZIL MSWINDOWS)
    DWORD dwStyle;
    FARPROC defaultCallback;

    void RegisterObject(char *className, char *winClassName, int *offset,
        FARPROC *procInstance, FARPROC *defProcInstance,
        ZIL_ICHAR *title = ZIL_NULLP(ZIL_ICHAR), HMENU menu = 0);
    #elif defined(ZIL OS2)
    ZIL_UINT32 flStyle;
    ZIL_UINT32 flFlag;
    PFNWP defaultCallback;

```

```

    ZIL_SCREENID RegisterObject(char *className, int *registeredClass,
        PFWNP *baseCallback, ZIL_ICHAR *title,
        void *controlData = ZIL_NULLP(void));
#elif defined(ZIL_MOTIF)
    static Arg args[50];
    static int nargs;
    void RegisterObject(WidgetClass widgetClass,
        ZIL_MOTIF_CONVENIENCE_FUNCTION convenienceFunction,
        EVENT_TYPE ccode, int useArgs = FALSE, int manage = TRUE,
        ZIL_SCREENID parent = 0);
#endif

    UI_WINDOW_OBJECT(int left, int top, int width, int height,
        WOF_FLAGS woFlags, WOAF_FLAGS woAdvancedFlags);
    EVENT_TYPE DrawBorder(ZIL_SCREENID screenID, UI_REGION &region,
        int fillRegion, EVENT_TYPE ccode);
    virtual EVENT_TYPE DrawItem(const UI_EVENT &event, EVENT_TYPE ccode);
    EVENT_TYPE DrawShadow(ZIL_SCREENID screenID, UI_REGION &region,
        int depth, int fillRegion, EVENT_TYPE ccode);
    EVENT_TYPE DrawText(ZIL_SCREENID screenID, UI_REGION &region,
        const ZIL_ICHAR *text, UI_PALETTE *palette, int fillRegion,
        EVENT_TYPE ccode);
    void Modify(const UI_EVENT &event);
    int NeedsUpdate(const UI_EVENT &event, EVENT_TYPE ccode);
    void RegisterObject(char *name);
    virtual void RegionMax(UI_WINDOW_OBJECT *object);

```

## General Members

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

- *repeatRate* is the time, in hundredths of seconds, that must elapse before an event is repeated. For example, when the down arrow on a `UIW_SCROLL_BAR` is depressed and held, scrolling events will occur at the rate specified by *repeatRate*.
- *doubleClickRate* is the time, in hundredths of seconds, that is used to determine if two consecutive mouse clicks are to be interpreted as a double click. If two mouse clicks occur within the time specified by *doubleClickRate*, then a double-clicked event is processed.
- *defaultStatus* is the status assigned to a window object when it is first constructed. If this value is changed to a valid status, all window objects will be created with this status (e.g., setting `WOS_GRAPHICS` would cause all window objects to be created with pixel boundaries and sizes).
- *display* is a pointer to the display class.
- *eventManager* is a pointer to the Event Manager.
- *windowManager* is a pointer to the Window Manager.

- *errorSystem* is a pointer to the error system. *errorSystem* should be initialized by the programmer at the beginning of the program if an error system is used.
- *helpSystem* is a pointer to the help system. *helpSystem* should be initialized by the programmer at the beginning of the program if a help system is used.
- *defaultStorage* is a pointer to the default storage system that is used for resource storage and/or retrieval. *defaultStorage* is used when loading icon and bitmap button images. It can also be used to load resources (i.e., UIW\_WINDOW). *defaultStorage* should be initialized by the programmer at the beginning of the program if it will be needed.
- *objectTable* is a table used to create objects from a persistent object file. The table contains entries for each type of object in the file. An entry consists of an object identifier and a pointer to the object's static **New()** function. When an object is being loaded from the file, its *searchID* is loaded which is then used to obtain the address of the appropriate **New()** function from *objectTable*. The **New()** function loads all the associated data for that object from the file. Each object that can be persistent must have a static **New()** member function. This is necessary because C++ does not allow the passing of non-static member functions and a constructor cannot be made static. The Designer creates an object table in the **.CPP** file it generates. When this file is compiled and linked into the application, it will be used to set the *objectTable* member. Alternately, a table can be created by hand and used to initialize *objectTable*.

```

static UI_ITEM _objectTable[] =
{
    {ID_BIGNUM, (UI_WINDOW_OBJECT*)VOIDF(UIW_BIGNUM::New), UIW_BIGNUM::_className, 0},
    {ID_BORDER, (UI_WINDOW_OBJECT*)VOIDF(UIW_BORDER::New), UIW_BORDER::_className, 0},
    {ID_BUTTON, (UI_WINDOW_OBJECT*)VOIDF(UIW_BUTTON::New), UIW_BUTTON::_className, 0},
    {ID_DATE, (UI_WINDOW_OBJECT*)VOIDF(UIW_DATE::New), UIW_DATE::_className, 0},
    {ID_GROUP, (UI_WINDOW_OBJECT*)VOIDF(UIW_GROUP::New), UIW_GROUP::_className, 0},
    {ID_HZ_LIST, (UI_WINDOW_OBJECT*)VOIDF(UIW_HZ_LIST::New), UIW_HZ_LIST::_className, 0},
    {ID_ICON, (UI_WINDOW_OBJECT*)VOIDF(UIW_ICON::New), UIW_ICON::_className, 0},
    {ID_PROMPT, (UI_WINDOW_OBJECT*)VOIDF(UIW_PROMPT::New), UIW_PROMPT::_className, 0},
    {ID_STRING, (UI_WINDOW_OBJECT*)VOIDF(UIW_STRING::New), UIW_STRING::_className, 0},
    {ID_TEXT, (UI_WINDOW_OBJECT*)VOIDF(UIW_TEXT::New), UIW_TEXT::_className, 0},
    {ID_TIME, (UI_WINDOW_OBJECT*)VOIDF(UIW_TIME::New), UIW_TIME::_className, 0},
    {ID_TITLE, (UI_WINDOW_OBJECT*)VOIDF(UIW_TITLE::New), UIW_TITLE::_className, 0},
    { ID_VT_LIST, (UI_WINDOW_OBJECT*)VOIDF(UIW_VT_LIST::New), UIW_VT_LIST::_className, 0 },
    {ID_WINDOW, (UI_WINDOW_OBJECT*)VOIDF(UIW_WINDOW::New), UIW_WINDOW::_className, 0},
    { ID_END, ZIL_NULLP(void), ZIL_NULLP(ZIL_ICHAR), 0 }
};
UI_ITEM *UI_WINDOW_OBJECT::objectTable = _objectTable;

```

**NOTE:** Initially, *objectTable* points to a default table (contained in **G\_JUMP.CPP**). The default table has all of the entries commented out to prevent unnecessary modules from linking into the application. If this table is needed, simply uncomment the required lines or copy and rename the table and uncomment the required lines.



- *userTable* is a table used to associate user functions and compare functions with the persistent objects that use them. The table contains entries for each function that appears in the persistent object file. An entry consists of a string used to identify the function and the address of the function. When an object is being loaded from the file, its *userFunctionName* or *compareFunctionName* is loaded which is then used to obtain the address of the appropriate user function or compare function address from *userTable*. If an object has a user function or compare function, they must be static. This is necessary because C++ does not allow the passing of non-static member functions. The Designer creates a user table in the **.CPP** file it generates. When this file is compiled and linked into the application, it will be used to set the *userTable* member. Alternately, a table can be created by hand and used to initialize *userTable*.

```
static UI_ITEM _userTable[] =
{
    { 0, VOIDF(FilenameCallback), "FilenameCallback", 0 },
    { 0, VOIDF(FilterCallback), "FilterCallback", 0 },
    { 0, VOIDF(DirectoryCompare), "DirectoryCompare", 0 },
    { ID_END, ZIL_NULLP(void), ZIL_NULLP(ZIL_ICHAR), 0 }
};
UI_ITEM *UI_WINDOW_OBJECT::userTable = _userTable;
```

**NOTE:** Initially, *userTable* points to a default table (contained in **G\_JUMP.CPP**). The default table is empty.

- *\_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 UI\_WINDOW\_OBJECT class, *\_className* is "UI\_WINDOW\_OBJECT."
- *eventMapTable* is a pointer to the event map table used by the object. The event map table is used to map raw events that were generated by an input device into logical events that the library objects can act upon. All objects use the same event map table by default, but individual objects can be assigned a special, user-defined table if behavior that is different than the default is required.
- *hotKeyMapTable* is a pointer to the hotkey map table used by the object. The hotkey map table is used to map the raw key event to the hotkey that was pressed. All objects use the same hotkey map table by default, but individual objects can be assigned a special, user-defined table if desired.
- *paletteMapTable* is a pointer to the palette map table used by the object. The palette map table is used to determine what color the object is supposed to be under particular circumstances. All objects use the same palette map table by default, but individual objects can be assigned a special, user-defined table if different colors are required.

- *screenID* is a unique identification given to a window object when it (or its parent) is attached to the Window Manager. This value is used in various places, but the most significant is when calling display functions. The *screenID* is used to identify the object and its display space. In most graphical operating systems, *screenID* is the window handle of the object. In DOS it is a value that identifies a screen region. In Motif it identifies the type of Motif widget the object is.
- *controlScreenID* is the Macintosh Handle to the object if it is a control object (e.g., button, scroll bar, etc.). Because the Macintosh toolbox has a specific handle type and specific functions for this type of object, a screenID specific to this type of object is required.
- *listScreenID* is the Macintosh Handle to the object if it is a list object (e.g., vertical list, etc.). Because the Macintosh toolbox has a specific handle type and specific functions for this type of object, a screenID specific to this type of object is required.
- *menuScreenID* is the Macintosh Handle to the object if it is a pop-up menu object. Because the Macintosh toolbox has a specific handle type and specific functions for this type of object, a screenID specific to this type of object is required.
- *textScreenID* is the Macintosh Handle to the object if it is an editable object (e.g., string, date, etc.). Because the Macintosh toolbox has a specific handle type and specific functions for this type of object, a screenID specific to this type of object is required.
- *windowScreenID* is the Macintosh pointer to the object if it is a window. Because the Macintosh toolbox has a specific pointer type and specific functions for this type of object, a screenID specific to this type of object is required.
- *woFlags* 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 window 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 applies to editable objects only.

**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" of *Programmer's Reference Volume 2* for information on changing DOS graphics mode styles and text mode styles.

**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 denoting the field as invalid, you force the user to enter an acceptable value. This flag applies to editable fields that have validation, such as dates and times.

**WOF\_JUSTIFY\_CENTER**—Center-justifies the text within the displayed object.

**WOF\_JUSTIFY\_RIGHT**—Right-justifies the text within the displayed 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. Setting this flag left-justifies the data, where applicable. 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 object. 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. Care should be taken when setting this flag on objects that do not use it by default, as undesired effects may result.

WOF\_UNANSWERED—Sets the initial status of the field to be "unanswered." An unanswered field is displayed as an empty field. This flag applies to editable objects only.

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. This flag applies to editable objects only.

- *woAdvancedFlags* are flags (common to all window objects) that determine the advanced operation of the window object. The following flags (declared in UI\_WIN.HPP) control the advanced operation of a window object:

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\_DRAG\_OBJECT—Allows the object or its contents to be used in a drag and drop operation.

WOAF\_LOCKED—Prevents the window object from being removed from the display. The WOAF\_LOCKED flag must be cleared before the window object can 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. This flag applies to objects attached directly to the Window Manager only.

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\_NO\_DESTROY**—Prevents the window from being destroyed when it is closed. If this flag is set, the window object can be removed from the display, but the programmer is responsible for destroying it. This flag applies to windows, parent or child, only.

**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 object from the keyboard. The object may still be selected using the mouse or a hotkey, but it will not become current.

**WOAF\_NORMAL\_HOT\_KEYS**—Allows the end-user to select an option by pressing its hotkey by itself, without the <Alt> key otherwise required for selecting with a hotkey.

**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\_OUTSIDE\_REGION**—Indicates the window object occupies space outside of the *trueRegion* region of the parent window but is still within the parent window (e.g., the UIW\_BORDER class).

**WOAF\_TEMPORARY**—Causes the object 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. This flag applies to objects attached directly to the Window Manager only.

- *woStatus* is a status flag that indicates the current state of a window object. The state may be reflected in the appearance of the object, its behavior or may only be recognized internally. These flags are updated occasionally when the object's state changes. The following status flags (declared in **UI\_WIN.HPP**) specify the window object's current status:

**WOS\_CHANGED**—Indicates that the window object's data has been modified by the end-user.

**WOS\_CURRENT**—Indicates that the window object is the current object in its parent's list. Only one window object in a list may have the **WOS\_CURRENT** flag set at any given time.

**WOS\_GRAPHICS**—Indicates that the window object region is specified in graphics coordinates as opposed to cell coordinates. This flag is set when an object's region is converted from cell coordinates to graphics coordinates.

**WOS\_INVALID**—Indicates that the window object's data is in an "invalid" state. An object's data is invalid if it is not within the absolute range for the object or is not within a range specified by the programmer.

**WOS\_MAXIMIZED**—Indicates that the window object is in a maximized state.

**WOS\_MINIMIZED**—Indicates that the window object is in a minimized state.

**WOS\_NO\_STATUS**—Indicates that the window object has no status.

**WOS\_OWNERDRAW**—Causes the window object's **DrawItem()** function to be called when the object needs to be drawn.

**WOS\_READ\_ERROR**—Indicates that there was an error reading a storage file.

**WOS\_REDISPLAY**—Indicates that the window object needs to be redisplayed.

**WOS\_SELECTED**—Indicates that the object is selected. The most common use for this flag is with buttons, where a button field can be in a selected or a non-selected state.

**WOS\_UNANSWERED**—Indicates that the window object's data is in an "unanswered" state.

- *trueRegion* is the region that is used to position the object in the operating system. An object's *trueRegion* region is calculated from its *relative* region in the **RegionMax()** function. In DOS, Curses and Macintosh, the *trueRegion* region is relative to the upper-left corner of the screen. In all other environments *trueRegion* is relative to the parent window's user region origin. The user region is the area of the window framed by, but not including, the support objects. In these environments, the *trueRegion* region is very similar to the *relative* region.
- *relative* is the region passed to the object in its constructor. It contains the desired position of the window object relative to the object's parent.
- *parent* is a pointer to the window object's parent. Its parent is the object it was attached to.
- *helpContext* is the help context identifier associated with the window object. The help context is passed to the help system when displaying help information. The help system displays the information identified by the help context. The help context value is generated by the Designer. The help context identifier name can be obtained from the header file produced by the Designer.
- *userFlags* is a flag field for the programmer's use. OpenZinc does not use this member. *userFlags* is maintained when the object is stored in a data file.
- *userStatus* is a status flag field for the programmer's use. OpenZinc does not use this member. As with other status flags, *userStatus* is not maintained when the object is stored in a data file.
- *userObject* is a pointer for the programmer's use. Since this is a void pointer, the object must be typecast by the programmer. This pointer is used by OpenZinc Application Framework for `UIW_COMBO_BOX` and `UIW_PULL_DOWN_MENU` objects and so should not be used by the programmer for these objects. For all other objects, the programmer is welcome to use this member as desired. If *userObject* has an entry in the `UI_WINDOW_OBJECT:.userTable`, then the text name in the table is saved in the data file.
- *userFunction* 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 is generally called by the library when the object becomes current, is selected or becomes non-current. 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<sub>out</sub>* 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<sub>in</sub>* 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<sub>in</sub>* is the run-time message passed to the object.

*ccode<sub>in</sub>* is the logical or system code that caused the user function to be called.

- *shell* is a pointer to the object's shell widget. It must be set before a call to **UI\_WINDOW\_OBJECT::RegisterObject( )** is made. This member is available for Motif only.
- *searchID* identifies the object's type. For example, if the object is a UIW\_BUTTON, its *searchID* is ID\_BUTTON. *searchID*, sometimes also referred to as the *objectID*, is stored with the object and is used to identify the object type when loading it from a .DAT file. A complete list of ZIL\_OBJECTID values can be found in **UI\_WIN.HPP**.
- *numberID* is a numerical value used to identify an object. OpenZinc places certain requirements on an object's *numberID* member. For example, a window's *numberID* must be greater than the *numberID* of all objects attached to the window. Thus, OpenZinc will modify a window's *numberID*. In general, this member should not be used by the programmer.
- *stringID* is a string name used to identify an object. The programmer is responsible for setting an object's *stringID*. *stringID* can be up to 32 characters, including the NULL terminator.
- *windowID* is an array that contains an object's inheritance hierarchy. This hierarchy is used at run-time by functions such as **MapEvent( )** and **MapPalette()**, neither of which have any knowledge of class hierarchies, to map events or palettes appropriately for the object. For example, for a UIW\_GROUP *windowID*[0] is



ID\_GROUP, windowID[1] is ID\_WINDOW, and the remaining entries are ID\_WINDOW\_OBJECT.

- *hotKey* is the character to use as the hotkey for the object. A value of 0 means that no hotkey is associated with the object.
- *font* is the ZIL\_LOGICAL\_FONT associated with the object, *font* is an index into the *fontTable* array that is a member of each display class. For more information regarding fonts, see the appropriate display chapter.
- *lastPalette* is a pointer to the last palette used to display the object. By maintaining a pointer, unnecessary mapping will be prevented.
- *userObjectName* is the string representation of the user object name. This variable is used for storage purposes only. The *userObjectName* is placed in the *objectTable* in an entry for the *userObject*.
- *userFunctionName* is the string representation of the user function name. This variable is used for storage purposes only. The *userFunctionName* is placed in the *userTable* in an entry for the *userFunction*.
- *clip* provides additional clip region information for an object, *clip* contains the area of the object clipped to the object's parent's user region. The user region is the region encompassed by, but not including, the support objects. Thus, if an object is partially outside its parents region (i.e., the window is clipping part of the object off) *clip* will prevent the object from drawing those parts of the object that are outside the parent.
- *pasteBuffer* is the global paste buffer. This member is available in DOS and Curses only.
- *pasteLength* is the length of the global paste buffer. This member is available in DOS and Curses only.
- *dwStyle* is the object's window style flags. This member is available in Windows only.
- *defaultCallback* is the base class default callback function (e.g., **DefWindowProc()** in Windows). This member is available in Windows, Windows NT and OS/2 only.
- *fStyle* is the object's window style flags. This member is available in OS/2 only.

- *flFlag* is the object's class-specific window style flags. This member is available in OS/2 only.
- *args* is an array of Xt resources. This member is available for Motif only.
- *nargs* is a counter of how many entries have been made in the *args* array. This member is available for Motif only.

NOTE: All the member functions in this chapter are advanced. In general, only derived window objects should need access to these functions.

## UI\_WINDOW\_OBJECT::UI\_WINDOW\_OBJECT

### Syntax

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

```
UI_WINDOW_OBJECT(int left, int top, int width, int height,  
                 WOF_FLAGS woFlags, WOAF_FLAGS woAdvancedFlags);
```

### 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 UI\_WINDOW\_OBJECT class object. A UI\_WINDOW\_OBJECT by itself is of little use, so the constructor is protected so that only derived classes can call it.

- *left<sub>in</sub>* and *top<sub>in</sub>* is the starting position of the object. Typically, these values are in cell coordinates. If the WOF\_MINICELL flag is set, however, these values will be interpreted as minicell values.
- *width<sub>in</sub>* is the width of the object. 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<sub>in</sub>* is the height of the object. 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* are flags (common to all window objects) that determine the general operation of the object. A full description of these flags is given at the beginning of this chapter.
- *woAdvancedFlags* are flags (common to all window objects) that determine the advanced operation of the object. A full description of these flags is given at the beginning of this chapter.

## Example

```
#include <ui_win.hpp>

UIW_BUTTON::UIW_BUTTON(int left, int top, int width, char *_string,
    USHORT _btFlags, USHORT _woFlags,
    void (*_userFunction)(void *object, UI_EVENT &event), USHORT _value) :
    UI_WINDOW_OBJECT(left, top, width, 1, _woFlags | WOFJBORDER,
        WOAF_NO_FLAGS),
    btFlags(_btFlags), btStatus(BTS_NO_STATUS), userFunction(_userFunction),
    string(NULL), value(_value), getString(NULL), time(0)
{
}
}
```

## UI WINDOW OBJECT::~~UI WINDOW OBJECT

### Syntax

```
#include <ui_win.hpp>

virtual ~UI_WINDOW_OBJECT(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 information associated with the UI\_WINDOW\_OBJECT class. This function is declared virtual so that the destructors associated with derived classes will be called before the base class destructor is called.

## UI\_WINDOW\_OBJECT::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<sub>out</sub>* is a pointer to the *\_className* member.

## UI\_WINDOW\_OBJECT::CreateMotifString

### Syntax

```
#include <ui_win.hpp>

static XmString CreateMotifString(ZIL_ICHAR *text,
    ZIL_ICHAR **displayText = ZIL_NULLP(ZIL_ICHAR *),
    int strip = 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 creates a Motif string. Motif strings have several components that must be associated with each other.

- *returnValue<sub>out</sub>* is the Motif string that is created.
- *text<sub>in</sub>* is the string that is to be placed in the Motif string.
- *displayText<sub>out</sub>* is a doubly indirected pointer to a string buffer. This pointer will be set to point to a buffer containing the *text*. The text will be the same as what was passed in unless *strip* is TRUE, in which case the text will have any hotkey designator characters removed.
- *strip<sub>in</sub>* specifies if hotkey designator characters (i.e., the '&' character) should be stripped from the string.

## UI\_WINDOW\_OBJECT::DrawBorder

### Syntax

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

```
EVENT_TYPE DrawBorder(ZIL_SCREENID screenID, UI_REGION &region,  
int fillRegion, 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 advanced function draws the border of a window object. The border drawn by this function is the thin border that results from the WOF\_BORDER flag being set. This border is not the UIW\_BORDER.

- *returnValue<sub>out</sub>* indicates if the border was drawn successfully. *returnValue* is TRUE if the border was drawn successfully. Otherwise, *returnValue* is FALSE.
- *screenID<sub>in</sub>* is the *screenID* of the object.
- *region<sub>in/out</sub>* is the region where the border should be drawn. This value is decremented by the size of the border. This argument should be a copy of the object's *trueRegion* region.
- *fillRegion<sub>in</sub>* specifies if the region within the border should be filled. If *fillRegion* is TRUE, the region will be filled. Otherwise, the region will not be filled.
- *ccode<sub>in</sub>* is the logical or system event that caused the border to be drawn.

## Example

```
#include <ui_win.hpp>
EVENT_TYPE UIW_WINDOW::Event(const UI_EVENT &event)
{
    // Switch on the event type.
    switch (ccode)
    {
        case S_CURRENT:
        case S_NON_CURRENT:
        case S_DISPLAY_ACTIVE:
        case {S_DISPLAY_INACTIVE:
            // Draw the border and fill the background.
            UI_WINDOW OBJECT: -.Event (event) ;
            if (!FlagSet(woStatus, WOS_REDISPLAY))
                break;
            UI_REGION region=trueRegion;
            display->VirtualGet(screenID, region);
            if (FlagSet(woFlags, WOF_BORDER) && trueRegion.Overlap(event.region))
                DrawBorder(screenID, region, FALSE, ccode);
        }
        // Return the control code,
        return (ccode);
    }
}
```

## UI\_WINDOW\_OBJECT::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 object. This allows the programmer to derive a new class from UI\_WINDOW\_OBJECT and handle the drawing of the object, if desired.

- *returnValue<sub>out</sub>* 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<sub>in</sub>* 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<sub>in</sub>* contains the logical interpretation of *event*.

## UI\_WINDOW\_OBJECT::DrawShadow

### Syntax

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

```
EVENT_TYPE DrawShadow(ZIL_SCREENID screenID, UI_REGION &region,  
int depth, int fillRegion, 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 advanced function draws the shadow that gives the object a three-dimensional appearance.

- *returnValue<sub>out</sub>* indicates if the shadow was drawn successfully. *returnValue* is TRUE if the shadow was drawn successfully. Otherwise, *returnValue* is FALSE.
- *screenID<sub>in</sub>* is the *screenID* of the object.
- *region<sub>in/out</sub>* is the region where the shadow should be drawn. This value is decremented by the size of the shadow. This argument should be a copy of the object's *trueRegion* region.
- *depth<sub>in</sub>* specifies the degree of shading to be drawn. Values greater than 0 (i.e., 1, 2) cause the object to appear to pop out of the screen, a value of 0 causes no shadow to be drawn, and values less than 0 (i.e., -1,-2) cause the object to appear depressed.
- *fillRegion<sub>in</sub>* specifies if the region within the shadow should be filled. If *fillRegion* is TRUE, the region will be filled. Otherwise, the region will not be filled.
- *ccode<sub>in</sub>* is the logical or system event that caused the shadow to be drawn.



## Example

```
#include <ui_win.hpp>

EVENT_TYPE UIW_BORDER::DrawItem(const UI_EVENT &, EVENT_TYPE ccode)
{
    // Check for text mode.
    if (display->isText)
    {
        UI_REGION region=parent->trueRegion;
        DrawShadow(screenID, region, 2, FALSE, ccode);
        return (ccode);
    }
}
```

## UI WINDOW OBJECT::DrawText

### Syntax

```
#include <ui_win.hpp>

EVENT_TYPE DrawText(ZIL_SCREENID screenID, UI_REGION &region,
    const ZIL_ICHAR *text, UI_PALETTE *palette, int fillRegion,
    EVENT_TYPE ccode);
```

### Portability

This function is available on the following environments:

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

### Remarks

This [advanced](#) function draws text in the object.

- *returnValue*<sub>out</sub> indicates if the text was drawn successfully. *returnValue* is TRUE if the text was drawn successfully. Otherwise, *returnValue* is FALSE.
- *screenID*<sub>in</sub> is the *screenID* of the object.

- *region<sub>in/out</sub>* is the region where the text should be drawn. This argument should be a copy of the object's *trueRegion* region.
- *text<sub>in</sub>* is a pointer to the text to be displayed. If the text string contains a hotkey character, denoted by a preceding '&' character, then it will be underlined if the application is running in graphics mode or highlighted if the application is running in text mode.
- *palette<sub>in</sub>* is a pointer to the palette structure that defines the color to draw the text. The palette's foreground color is used to draw the text. The palette's background color is used to draw the background of the text (if *fillRegion* is TRUE).
- *fillRegion<sub>in</sub>* specifies if the region within the text should be filled. If *fillRegion* is TRUE, the region will be filled. Otherwise, the region will not be filled.
- *ccode<sub>in</sub>* is the logical or system event that caused the text to be drawn.

## UI\_WINDOW\_OBJECT::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 object. It is declared virtual so that any derived class can override its default operation. This function processes general messages for all objects. If a derived object does not handle a particular message in its **Event()** function, it should pass the event to its base class. Thus, events that are not handled by derived objects may eventually be passed to this function since all objects are derived from UI\_WINDOW\_OBJECT.

- *returnValue<sub>out</sub>* 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<sub>m</sub>* contains a run-time message for the 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. This message is interpreted from a keyboard 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**—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.

**L\_HELP**—Requests the context-sensitive help associated with the object.

**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\_NEXT**—This message is passed to the object's parent, if one exists, for processing.

**L\_PREVIOUS**—This message is passed to the object's parent, if one exists, for processing.

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

**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 object. This message allows the object to alter the mouse image.

**S\_ADD\_OBJECT**—This message is passed to the object's parent, if one exists, for processing.

**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\_CLOSE**—If this message is received at the `UI_WINDOW_OBJECT` level, it will have come from the operating system. The message is placed on the event queue for processing.

**S\_CLOSE\_TEMPORARY**—If this message is received at the `UI_WINDOW_OBJECT` level, it will have come from the operating system. The message is placed on the event queue for processing.

**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 that the end-user is dragging the object for a copy operation.

**S\_DRAG\_MOVE\_OBJECT**—Indicates that the end-user is dragging the object for a move operation.

**S\_DROP\_COPY\_OBJECT**—Indicates that the end-user is dropping an object for a copy operation. The dragged object's text is copied to this object.

**S\_DROP\_MOVE\_OBJECT**—Indicates that the end-user is dropping an object for a move operation. The dragged object's text is moved to this 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\_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 or object.

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\_SIZE—Causes the object to change its size. The object's new *relative* region is passed in *event.region*.

S\_SUBTRACT\_OBJECT—This message is passed to the object's parent, if one exists, for processing.

All other events cause the S\_UNKNOWN message to be returned.

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.

## Example

```
#include <ui_win.hpp>

EVENT_TYPE UIW_BUTTON::Event(const UI_EVENT &event)
{
    EVENT_TYPE ccode = UI_WINDOW_OBJECT::LogicalEvent(event, ID_BUTTON);
    switch (ccode)
    {
        case S_CREATE:
        case S_SIZE:

            default:
                ccode = UI_WINDOW_OBJECT::Event(event);
                break;
    }
}
```

```

    // Return the control code,
    return (ccode);
}

```

## **UI WINDOW OBJECT::Font**

### **Syntax**

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

```
ZIL_LOGICAL_FONT Font(ZIL_LOGICAL_FONT font = FNT_NONE);
```

### **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 logical font for an object or returns the logical font in use by the object. See the individual display class chapters for more information regarding fonts.

- *returnValue*<sub>out</sub> is the logical font in use by the object.
- *font*<sub>in</sub> is the logical font to be assigned to the object. If *font* is FNT\_NONE, the default, then the font is not changed but the current font will be returned.

## **UI WINDOW OBJECT::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 in the object's list. They do a depth-first search of the objects in the list, searching for a match on the identification data specified. If the object is not derived from UI\_LIST, no action is performed.

The first overloaded function returns the object whose *stringID* matches *name*.

- *returnValue<sub>out</sub>* is a pointer to the object whose *stringID* matches *name*. If no object matches *name*, NULL is returned.
- *name<sub>in</sub>* is the *stringID* of the object to be located.

The second function returns the object whose *numberID* matches *\_numberID*.

- *returnValue<sub>out</sub>* is a pointer to the object whose *numberID* matches *\_numberID*. If no object matches *\_numberID*, NULL is returned.

*numberID<sub>in</sub>* is the *numberID* of the object to be located.

## UI\_WINDOW\_OBJECT::HotKey

### Syntax

```
#include <ui_win.hpp>

unsigned HotKey(unsigned hotKey = 0);
    or
unsigned HotKey(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

These overloaded functions set the hotkey for the object. If an object added to a window contains sub-objects with hotkeys, then the object should have its hotkey set to `HOT_KEY_SUB_WINDOW` so that its children can process hotkeys. For example, a `UIW_TOOL_BAR` with buttons on it should have its hotkey set to `HOT_KEY_SUB_WINDOW`.

The first overloaded function sets the hotkey for the object.

- *returnValue<sub>out</sub>* is the value of the hotkey after it has been changed.
- *hotKey<sub>in</sub>* is the new hotkey value. Any alphanumeric character can be used for a hotkey. If *hotKey* is 0, no change is made but the hotkey value is returned.

The second overloaded function sets the hotkey for the object by parsing the text that is passed in looking for the hotkey designator character (an '&' by default).

- *returnValue<sub>out</sub>* is the value of the hotkey after it has been changed.
- *text<sub>in</sub>* is a pointer to the text for the object. This text is searched for the hotkey designator character (an '&' by default). If the character is found, then the character immediately after it is set to be the object's hotkey.

## Example

```
ExampleFunction(UI_WINDOW_OBJECT *object1, UI_WINDOW_OBJECT *object2)
{
    object1->HotKey('A');

    unsigned value = object1->HotKey( );
    object2->HotKey(value);
}
```

## UI\_WINDOW\_OBJECT::Information

### Syntax

```
#include <ui_win.hpp>

virtual void *Information(INFORMATION_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*<sub>out</sub> 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*<sub>in</sub> 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 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, for example, the object is a button and 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, for example, the object is a button and 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, for example, the object is a button and 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\_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, for example, the object is a button and the BTF\_FLAGS are desired, *objectID* should be ID\_BUTTON. 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. Programmers should use a window's *numberID* with caution as it may change at run-time. For more details, see the note accompanying the description of **UI\_WINDOW\_OBJECT::NumberID( )** in this chapter.

**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, for example, the object is a button and the `BTS_STATUS` is desired, *objectID* should be `ID_BUTTON`. This allows the object to process the request at the proper level.

**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, for example, the object is a button and 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, for example, the object is a button and 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.

- *data<sub>in/out</sub>* 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<sub>in</sub>* 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>

void *UIW_BUTTON::Information(ZIL_INFO_REQUEST request, void *data,
                             ZIL_OBJECTID ObjectID)
{
    // Switch on the request.
    switch (request)
    {

        case I_GET_FLAGS:
        case I_SET_FLAGS:
        case I_CLEAR_FLAGS:
            if (ObjectID && ObjectID != ID_BUTTON)
                data = UI_WINDOW_OBJECT::Information(request, data, ObjectID);
            else if (request == I_GET_FLAGS && !data)
                data = &btFlags;
            else if (request == I_GET_FLAGS)
                *(BTF_FLAGS *)data = btFlags;
            else if (request == I_SET_FLAGS)
                btFlags |= *(BTF_FLAGS *)data;
            else
                btFlags &= ~(*(BTF_FLAGS *)data);
            break;

    }
    // Return the information,
    return (data);
}
```

## UI\_WINDOW\_OBJECT::Inherited

### Syntax

```
#include <ui_win.hpp>

int Inherited(ZIL_OBJECTID matchID);
```

## Portability

This function is available on the following environments:

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

## Remarks

This function indicates if the object is inherited from a particular class, specified by the `objectID`.

- `returnValueout` is TRUE if the window object is inherited from the class specified by `matchID`. Otherwise, `returnValue` is FALSE.
- `matchIDin` is the `objectID` to match. If the object is derived from the class specified by `matchID`, `returnValue` will be TRUE. `Inherited()` determines the object's inheritance hierarchy by inspecting the object's `windowID` array.

## Example

```
#include <ui_win.hpp>

EVENT_TYPE UIW_ICON::Event(const UI_EVENT &event)
{

    // Switch on the event type.
    EVENT_TYPE ccode = event.type;
    switch (ccode)
    {

    case S_CREATE:
        if (!_iconJumpInstance)
            _iconJumpInstance = (FARPROC)IconJumpProcedure;
        UI_WINDOW_OBJECT::Event(event);
        if (parent->Inherited(ID_LIST))
            parent->woStatus |= WOS_OWNERDRAW;

    }

    // Return the control code,
    return (ccode);
}
```

## UI\_WINDOW\_OBJECT::Logical Event

### Syntax

```
#include <ui_win.hpp>

EVENT_TYPE LogicalEvent(const UI_EVENT &event, ZIL_OBJECTID currentID = 0,
    int nativeType = TRUE);
```

### 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 by all window objects to interpret a raw event.

- *returnValue<sub>out</sub>* is the logical event that is interpreted from the raw event and object identification.
- *event<sub>in</sub>* is the raw event that is to be interpreted. Typically, this event was generated by a device, such as the keyboard.
- *currentID<sub>in</sub>* is the objectID of the object interpreting the event. This value is used to determine the mapping of a logical event.
- *nativeType<sub>in</sub>* specifies if the message is to be processed as a native operating system message. By default, most keyboard and mouse events are not translated in graphical operating systems. If *nativeType* is TRUE, this is how processing will occur and the logical event that is returned will indicate that the event is a native operating system event (e.g., a Windows event will return E\_MSWINDOWS). If *nativeType* is FALSE and the event did not directly map to a logical event, however, keyboard events will return E\_KEY and mouse events will return E\_MOUSE, no matter what operating system the application is running on. The native event is still returned in the *message* field of the UI\_EVENT structure.

## Example

```
#include <ui_win.hpp>

EVENT_TYPE UIW_BORDER::Event(const UI_EVENT &event)
{
    // Switch on the event type.
    UI_REGION region;
    EVENT_TYPE ccode = LogicalEvent(event, ID_BORDER);
    switch (ccode)

// Return the control code,
return (ccode);
```

## UI WINDOW OBJECT::LogicalPalette

### Syntax

```
#include <ui_win.hpp>

UI_PALETTE *LogicalPalette(LOGICAL_EVENT logicalEvent,
    ZIL_OBJECTID currentID = 0);
```

### 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 by all window objects to determine which palette should be used to draw the object based on the logical event.

- *returnValue<sub>out</sub>* is a pointer to the palette that should be used to draw the object.
- *logicalEvent<sub>in</sub>* is the logical event that determines which palette entry to use. For example, if *logicalEvent* is S\_CURRENT and the object is current, then the PM\_CURRENT palette will be used.



- *currentID<sub>m</sub>* is the objectID of the object interpreting the event. This value is used to determine the palette mapping given the logical event.

## Example

```
#include <ui_win.hpp>

EVENT_TYPE EXAMPLE_CLASS::Event(const UI_EVENT &event)
{
    // Switch on the event type.
    UI_REGION region;
    EVENT_TYPE ccode = UI_WINDOW_OBJECT::LogicalEvent(event, ID_WINDOW_OBJECT);
    switch (ccode)
    {
        case S_DISPLAY_INACTIVE:
        case S_DISPLAY_ACTIVE:
            // Draw the borders around the object.
            UI_WINDOW_OBJECT::Event(event);
            UI_PALETTE *palette = LogicalPalette(ccode, ID_WINDOW_OBJECT);

            break;

    }

    // Return the control code,
    return (ccode);
}
```

## UI\_WINDOW\_OBJECT::Modify

### Syntax

```
#include <ui_win.hpp>

void Modify(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 is used to change an object's size or position. When this function is called, an XOR outline of the object appears. The outline can be moved or sized with the arrow keys on the keyboard or by moving the mouse (if **Modify()** was invoked as a response to a mouse click). When <Enter> is pressed or the mouse button is released, the object will take on its new size or position.

- *event<sub>m</sub>* contains the type of modification to be done, *event*'s members are set to the following values:

*event.type* specifies if the function is to size (*event.type* is L\_SIZE) or move (*event.type* is L\_MOVE) the object.

*event.rawCode* specifies which edges of the object can be modified (i.e., M\_LEFT\_CHANGE, M\_TOP\_CHANGE, M\_RIGHT\_CHANGE or M\_BOTTOM\_CHANGE.)

## Example

```
#include <ui_win.hpp>

EVENT_TYPE UIW_WINDOW::Event(const UI_EVENT &event)
{
    UI_WINDOW_OBJECT *object;

    // Switch on the event type.
    switch (cocode)
    {
        case S_MOVE:
        case S_SIZE:
            Modify(event);
            break;
    }
}
```

## UI\_WINDOW\_OBJECT::NeedsUpdate

### Syntax

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

```
int NeedsUpdate(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 advanced function determines if the object needs to be redrawn. Currently, this function always returns TRUE.

- *returnValue*<sub>out</sub> is always TRUE.
- *event*<sub>in</sub> is not used.
- *ccode*<sub>in</sub> is not used.

## UI\_WINDOW\_OBJECT::Next

### Syntax

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

```
UI_WINDOW_OBJECT *Next(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 next object attached to the object's parent, if one exists.

## UI\_WINDOW\_OBJECT::NumberID

### Syntax

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

```
ZIL_NUMBERID NumberID(ZIL_NUMBERID numberID = 0);
```

## 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 or retrieves an object's *numberID*.

**NOTE:** The variable *numberID* is used by the library. An object's *numberID* must be unique among all objects on a window and a window's *numberID* must be greater than any of its children's *numberID*'s. Thus, for a window, *numberID* may be modified by the library. For this reason, *numberID* should not be used to identify a window. It is recommended that programmers use *stringID*, rather than *numberID*, to identify objects.

- *returnValue<sub>out</sub>* is the object's *numberID*.

- *numberID<sub>in</sub>* is the new *numberID* for the object. If this value is 0, the object's *numberID* is not modified, but its current *numberID* is returned.

## UI\_WINDOW\_OBJECT::Previous

### Syntax

```
#include <ui_win.hpp>

UI_WINDOW_OBJECT *Previous(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 previous object attached to the object's parent, if one exists.

## UI\_WINDOW\_OBJECT::RedisplayType

### Syntax

```
#include <ui_win.hpp>

EVENT_TYPE RedisplayType(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 indicates what type of redisplay the object requires. The object may need to be drawn to appear current, active or inactive.

- *returnValue<sub>out</sub>* indicates how the object needs to be redrawn. *returnValue* can have one of the following values:

**S\_CURRENT**—Indicates that the object should be drawn to appear current.

**S\_DISPLAY\_ACTIVE**—Indicates that the object should be drawn to appear active.

**S\_DISPLAY\_INACTIYE**—Indicates that the object should be drawn to appear inactive.

## UI\_WINDOW\_OBJECT::RegionConvert

### Syntax

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

```
void RegionConvert(UI_REGION &region, int absolute);
```

### Portability

This function is available on the following environments:

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

## Remarks

This function converts a region to pixel coordinates. When an object is constructed, it is usually given cell coordinates for its size and position. This function is used to convert the coordinates into pixel coordinates when in graphics mode.

- *region<sub>in/out</sub>* is the region that is to be converted.

- *absolute<sub>m</sub>* specifies if the function should add a *preSpace* and *postSpace* region to the converted region. If *absolute* is TRUE, the converted region will not have a *preSpace* and *postSpace* region added. Otherwise, *preSpace* and *postSpace* will be added to the converted region. *preSpace* and *postSpace* are members of the display class that define the space between the top or bottom of a cell and the top or bottom of the object itself.

## Example

```
#include <ui_win.hpp>

EVENT_TYPE UI_WINDOW_OBJECT::Event(const UI_EVENT &event)
{
    static ZIL_TIME lastTime;
    UI_WINDOW_OBJECT *object;

    // Switch on the event type.
    EVENT_TYPE ccode = LogicalEvent(event);
    switch (ccode)
    {
        case S_INITIALIZE:

            dwStyle |= parent ? WS_CHILD | WS_VISIBLE : WS_OVERLAPPED;
            RegionConvert(relative, (parent && !FlagSet(woFlags,
                WOF_NON_FIELD_REGION)) ? FALSE : TRUE);
            break;

    }
}
}
```

## UI\_WINDOW\_OBJECT::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 sets the object's *trueRegion* and *clip* regions. They are set to the object's *relative* region.

- *object<sub>in</sub>* is a pointer to the object that needs its *trueRegion* region assigned. Its *trueRegion* member will be modified with its actual position.

## Example

```
#include <ui_win.hpp>

#if defined(ZIL_MSWINDOWS)
EVENTJTYPE UIW_PULL_DOWN_MENU::Event(const UI_EVENT &event)
{
    UI_WINDOW_OBJECT *object;

    // Switch on the event type.
    EVENT_TYPE ccode = event.type;
    switch (ccode)

    case S_SIZE:
        // Compute the positions of the window objects.
        if (FlagSet(pdStatus, PDS_MAIN_MENU))
        {
            true.top = true.bottom = 0;
            break;
        }
        else if (ccode == S_SIZE)
            parent->RegionMax(this);

        break;

    }
    return (ccode);
}
```



## UI\_WINDOW\_OBJECT::RegisterObject

### Syntax

```
#include <ui_win.hpp>

void RegisterObject(char *className, char *winClassName,
    WNDPROC *defProcInstance, ZIL_ICHAR *title = ZIL_NULLP(ZIL_ICHAR),
    HMENU menu = 0);
    or
void RegisterObject(char *className, char *winClassName, int *offset,
    FARPROC *procInstance, FARPROC *defProcInstance,
    ZIL_ICHAR *title = ZIL_NULLP(ZIL_ICHAR), HMENU menu = 0);
    or
ZIL_SCREENID RegisterObject(char *className, PSZ os2ClassName,
    int *classRegistered, ZIL_ICHAR *title, void *controlData = ZIL_NULLP(void));
    or
void RegisterObject(WidgetClass widgetClass,
    ZIL_MOTIF_CONVENIENCE_FUNCTION convenienceFunction,
    EVENT_TYPE ccode, int useArgs = FALSE, int manage = TRUE,
    ZIL_SCREENID parent = 0);
    or
void RegisterObject(char *name)\
```

### Portability

This function is available on the following environments:

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

### Remarks

These functions register the object with the operating system. An object will not receive any messages from the operating system if it has not been registered.

The first function is specific to Windows NT.

- *className<sub>m</sub>* is the name of the OpenZinc object being registered. These are OpenZinc names such as "UIW\_VT\_LIST," "UIW\_BUTTON," "UIW\_WINDOW," etc.

- *winClassName<sub>m</sub>* is the name of the Windows class from which the object being registered is derived. *winClassName* is the base class of the Windows NT object and does not necessarily follow OpenZinc's class hierarchy. For example, UIW\_VT\_LIST is derived from UIW\_WINDOW, but in the Windows environment, the OpenZinc object is derived from LIST\_BOX and must be registered that way.
- *defProcInstance<sub>out</sub>* is the address of the default callback function that Windows provides for each object.
- *title<sub>m</sub>* is a string containing the title of the window, if any.
- *menu<sub>m</sub>* is the pull-down menu associated with the window, if any.

The second function is specific to Windows.

- *className<sub>in</sub>* is the name of the OpenZinc object being registered. These are OpenZinc names such as "UIW\_VT\_LIST," "UIW\_BUTTON," "UIW\_WINDOW," etc.
- *winClassName<sub>m</sub>* is the name of the Windows class from which the object being registered is derived. *winClassName* is the base class of the Windows object and does not necessarily follow OpenZinc's class hierarchy. For example, UIW\_VT\_LIST is derived from UIW\_WINDOW, but in the Windows environment, the OpenZinc object is derived from LIST\_BOX and must be registered that way.
- *offset<sub>in/out</sub>* is the size (in bytes) of the user space that accompanies messages from the operating system. Initially, this value should be -1 which will cause the function to initialize the object.
- *procInstance<sub>in/out</sub>* is the address of the callback function that Windows will call when the object gets an event. This function is provided, for each object, by OpenZinc Application Framework.
- *defProcInstance<sub>in/out</sub>* is the address of the default callback function that Windows provides for each object.
- *title<sub>m</sub>* is a string containing the title of the window, if any.
- *menu<sub>m</sub>* is the pull-down menu associated with the window, if any.

The third function is specific to OS/2.

- *returnValue<sub>out</sub>* is the ZIL\_SCREENID of the object created.

- *className<sub>in</sub>* is the name of the OpenZinc object being registered. These are OpenZinc names such as "UIW\_VT\_LIST," "UIW\_BUTTON," "UIW\_WINDOW," etc.
- *classRegistered<sub>in/out</sub>* when passed in, indicates if the function should attempt to register the class. If *classRegister* is TRUE, the function will try to register to the class with the OS/2 operating system. Otherwise, it will only create an instance of the object. *classRegister* is modified by the function to indicate if the class was registered.
- *baseCallback<sub>in/out</sub>* is the address of the default callback function that OS/2 provides for each object.
- *title<sub>in</sub>* is a string containing the title of the window, if any.
- *controlData<sub>in</sub>* is frame control data. The *flFlag* member is used for this value.

The fourth function is specific to Motif.

- *widgetClass<sub>in</sub>* is the type of Xt widget that is to be created. If this parameter is used, the *convenienceFunction* parameter should be NULL.
- *convenienceFunction<sub>in</sub>* is the convenienceFunction that is to be used to create the object. If this parameter is used, the *widgetClass* parameter should be NULL.
- *ccode<sub>in</sub>* distinguishes the circumstances under which **RegisterObject()** is being called (i.e., **RegisterObject()** could be called with an S\_SIZE, an S\_CREATE or some other ccode. This parameter allows the function to distinguish between the various events).
- *useArgs<sub>m</sub>* specifies whether the args array has been filled at all prior to the call to **RegisterObject()**. If *useArgs* is TRUE, then the array has been partially filled already.
- *manage<sub>in</sub>* indicates if the widget created by the corresponding convenience function should be managed. Since most convenience functions don't manage their widget by default, setting *manage* to TRUE will cause the widget to be managed.
- *parent<sub>in</sub>* specifies the Xt parent of the object.

The fifth function causes the object to be registered by sending an S\_REGISTER\_OBJECT message to the object.

- *name<sub>in</sub>* is the class name of the object to be registered.

## Example

```
EVENT_TYPE UIW_VT_LIST::Event(const UI_EVENT &event)
{
    UI_WINDOW_OBJECT *object;

    // Switch on the event type.
    EVENT_TYPE ccode = LogicalEvent(event);
    switch (ccode)

    case S_CREATE:
        UI_WINDOW_OBJECT::Event(event);
        RegisterObject("UIW_VT_LIST", "LISTBOX", &_listOffset,
            &_listJumpInstance, &_listCallback, NULL);
        SendMessage(screenID, WM_SETREDRAW, FALSE, 0);
        for (object = First(); object; object = object->Next())
            object->Event(event);
        SendMessage(screenID, WM_SETREDRAW, TRUE, 0);
        break;

    }

    // Return the control code,
    return (ccode);
}
```

## UI\_WINDOW\_OBJECT::Root

### Syntax

```
#include <ui_dsp.hpp>
```

```
UI_WINDOW_OBJECT *Root(int mdiChild = 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 returns a pointer to the object's root window.

- *returnValue<sub>out</sub>* is a pointer to the object's root window. The root window is the window that is attached to the Window Manager, unless *mdiChild* is TRUE, in which case the root window is the MDI child window that is attached to the MDI parent window.
- *mdiChild<sub>in</sub>* specifies if the root window that is returned should be the top-most root or the MDI child window root. If *mdiChild* is TRUE, the MDI child root is returned. Otherwise, the root window, attached to the Window Manager, is returned.

## UI\_WINDOW\_OBJECT::SearchID

### Syntax

```
#include <ui_win.hpp>

ZIL_OBJECTID SearchID(void);
```

### Portability

This function is available on the following environments:

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

### Remarks

This member function returns the *searchID* of the object. The *searchID*, sometimes also referred to as the *objectID*, identifies the object's type. For example, a button has a *searchID* of ID\_BUTTON. The possible values for *searchID* are defined in UI\_WINDOW.HPP.

- *returnValue<sub>out</sub>* is the object's *searchID*.

### Example

```
#include <ui_win.hpp>

ExampleFunction1(UIW_WINDOW *window)
```

```

        ZIL_OBJECTID searchID = window->SearchID();

    }

```

## UI\_WINDOW\_OBJECT::StringID

### Syntax

```

#include <ui_win.hpp>

ZIL_ICHAR *StringID(const ZIL_ICHAR *stringID = 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 member function sets or obtains the *stringID* of the object.

- *returnValue<sub>out</sub>* is the object's *stringID*.
- *stringID<sub>in</sub>* is the new *stringID* for the object. If this value is NULL, the object's *stringID* is not modified, but its current *stringID* is returned.

### Example

```

#include <ui_win.hpp>
UIW_WINDOW *ExampleFunction1(void)
{
    // Create the standard Hello World! window.
    UIW_WINDOW *window = UIW_WINDOW::Generic(2, 2, 40, 6, "Hello World Window");
    window->StringID("HELLO_WORLD_WINDOW");
    // Add the window objects to the window.
    *window
        + new UIW_TEXT(0, 0, 0, 0, "Hello, World!", 256,
            WNF_NO_FLAGS, WOF_NON_FIELD_REGION);
    // Return a pointer to the window,
    return (window);
}

```

## UI\_WINDOW\_OBJECT::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 *screenID* of the object's top-most Motif Widget if the object is made up of multiple Widgets. Some objects may be created as a combination of several Widgets with one Widget acting as the top-most, controlling Widget. This function obtains that Widget's *screenID*.

- *returnValue<sub>out</sub>* is the *screenID* of the top-most Widget.

## UI\_WINDOW\_OBJECT::UserFunction

### Syntax

```
#include <ui_win.hpp>

EVENT_TYPE UserFunction(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 function calls the object's user function, if it exists, or else validates the window object. When a window object receives the L\_SELECT, S\_CURRENT or S\_NON\_CURRENT messages, it will call **UserFunction()**. **UserFunction()** calls the object's **userFunction()** if it exists. Otherwise it calls the object's **Validate()** function.

- *returnValue<sub>out</sub>* is the return value from the user function or validation function.
- *event<sub>in</sub>* is the event that caused **UserFunction()** to be called, *event* is passed to the user function.
- *ccode<sub>in</sub>* is the logical event that was interpreted from the event. It is used to determine the type of action that is to take place.

## UI\_WINDOW\_OBJECT::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 L\_SELECT 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. The implementation of **Validate()** at the UI\_WINDOW\_OBJECT level is merely as a stub. Not all objects can be validated (e.g., it doesn't make sense to validate a button's data). Those objects that can be validated have an overloaded



**Validate()** function. Those objects that cannot be validated, though, do not have an overloaded **Validate()**. This function is called in those cases.

- *returnValue<sub>out</sub>* is always 0, indicating success.
- *processError<sub>in</sub>* is not used.

## Storage Members

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

## UI\_WINDOW\_OBJECT::UI\_WINDOW\_OBJECT

### Syntax

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

```
UI_WINDOW_OBJECT(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 `UI_WINDOW_OBJECT` by loading the object from a data file. Typically, the programmer does not need to use this constructor. If an 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<sub>in</sub>* is the name of the object to be loaded.

- *file<sub>in</sub>* 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."
- *object<sub>in</sub>* 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."
- *objectTable<sub>in</sub>* 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 this chapter. 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<sub>in</sub>* 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 this chapter. 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.

## UI\_WINDOW\_OBJECT::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 `UI_WINDOW_OBJECT` from a persistent object data file. It is called by the persistent constructor and is typically not used by the programmer.

- *name<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* 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`."
- *object<sub>in</sub>* 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`."
- *objectTable<sub>in</sub>* 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 this chapter. 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<sub>in</sub>* 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 this chapter. 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.

## UI\_WINDOW\_OBJECT::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<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* 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."
- *object<sub>in</sub>* 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."
- *objectTable<sub>in</sub>* 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 *UIWINDOWOBJECT::objectTable* in this chapter. 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<sub>in</sub>* 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 *UIWINDOWOBJECT::userTable* in this chapter. 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.

## **UI WINDOW OBJECT::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
- Macintosh • OSF/Motif • Curses
- OS/2
- NEXTSTEP

### **Remarks**

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

- *returnValue<sub>out</sub>* is a pointer to the object's **New()** function.

## **UI WINDOW OBJECT::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<sub>in</sub>* is the name of the object to be stored.
- *file<sub>in</sub>* 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."
- *object<sub>in</sub>* 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."
- *objectTable<sub>in</sub>* 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 this chapter. 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<sub>in</sub>* 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 this chapter. 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 44 - UI\_XT\_DISPLAY

The `UI_XT_DISPLAY` class implements a graphics display that uses the X Toolkit Intrinsics and Xlib graphics functions to draw to the screen. This display class is used with Motif applications. Since the `UI_XT_DISPLAY` class is derived from `UI_DISPLAY`, only details specific to the `UI_XT_DISPLAY` class are given in this chapter. For descriptions and examples regarding virtual or inherited display members, see "Chapter 7—`UI_DISPLAY`."

Applications using the `UI_XT_DISPLAY` class are true X Window or OSF/Motif programs. The X resource database is used to specify such resources as default colors, widget fonts, etc. While the default application class name is "ZincApp", users can create their own class names and files that specify the application defaults. The X foreground and background resources currently override `UI_WINDOW_OBJECT::paletteMapTable` for objects. The `UI_WINDOW_OBJECT::paletteMapTable` is used only for graphics display primitives. The X resource file is also used to specify which fonts are used as the default OpenZinc fonts. If a different font is desired, simply make the appropriate changes in the X resource file.

The `UI_XT_DISPLAY` class is declared in `UI_DSP.HPP`. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UI_XT_DISPLAY : public UI_DISPLAY
{
public:
    struct XFONt
    {
        XFontStruct *fontStruct;
        XmFontList fontList;
#ifdef ZIL_UNICODE
        XFontSet fontSet;
#endif
    };
    static XFONt fontTable[ZIL_MAXFONTS];

    UI_XT_DISPLAY(int *argc = ZIL_NULLP(int),
                 char **argv = ZIL_NULLP(char *), char *appClass = "ZincApp",
                 XrmOptionDescList options = ZIL_NULLP(XrmOptionDescRec),
                 Cardinal numOptions = 0,
                 String *fallbackResources = ZIL_NULLP(String));
    virtual ~UI_XT_DISPLAY(void);
    virtual void Bitmap(ZIL_SCREENID screenID, int column, int line,
                      int bitmapWidth, int bitmapHeight, const ZIL_UINT8 *bitmapArray,
                      const UI_PALETTE *palette = ZIL_NULLP(UI_PALETTE),
                      const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
                      ZIL_BITMAP_HANDLE *colorBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE),
                      ZIL_BITMAP_HANDLE *monoBitmap = ZIL_NULLP(ZIL_BITMAP_HANDLE));
    virtual void BitmapArrayToHandle(ZIL_SCREENID screenID, int bitmapWidth,
                                     int bitmapHeight, const ZIL_UINT8 *bitmapArray,
                                     const UI_PALETTE *palette, ZIL_BITMAP_HANDLE *colorBitmap,
                                     ZIL_BITMAP_HANDLE *monoBitmap);
    virtual void BitmapHandleToArray(ZIL_SCREENID screenID,
```



```

        ZIL_BITMAP_HANDLE eolorBitmap, ZIL_BITMAP_HANDLE monoBitmap,
        int *bitmapWidth, int *bitmapHeight, ZIL_UINT8 **bitmapArray);
virtual void Ellipse(ZIL_SCREENID screenID, int column, int line,
        int startAngle, int endAngle, int xRadius, int yRadius,
        const UI_PALETTE *palette, int fill = FALSE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void IconArrayToHandle(ZIL_SCREENID screenID, int iconWidth,
        int iconHeight, const ZIL_UINT8 *iconArray,
        const UI_PALETTE *palette, ZIL_ICON_HANDLE *icon);
virtual void IconHandleToArray(ZIL_SCREENID screenID,
        ZIL_ICON_HANDLE icon, int *iconWidth, int *iconHeight,
        ZIL_UINT8 **iconArray);
virtual void Line(ZIL_SCREENID screenID, int column1, int line1,
        int column2, int line2, const UI_PALETTE *palette, int width = 1,
        int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual ZIL_COLOR MapColor(const UI_PALETTE *palette, int isForeground);
virtual void Polygon(ZIL_SCREENID screenID, int numPoints,
        const int *polygonPoints, const UI_PALETTE *palette,
        int fill = FALSE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void Rectangle(ZIL_SCREENID screenID, int left, int top,
        int right, int bottom, const UI_PALETTE *palette, int width = 1,
        int fill = FALSE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void RectangleXORDiff(const UI_REGION &oldRegion,
        const UI_REGION &newRegion, ZIL_SCREENID screenID = ID_SCREEN,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION));
virtual void RegionDefine(ZIL_SCREENID screenID, int left, int top,
        int right, int bottom);
virtual void RegionMove(const UI_REGION &oldRegion, int newColumn,
        int newLine, ZIL_SCREENID oldScreenID = ID_SCREEN,
        ZIL_SCREENID newScreenID = ID_SCREEN);
virtual void Text(ZIL_SCREENID screenID, int left, int top,
        const ZIL_ICHAR *text, const UI_PALETTE *palette, int length = -1,
        int fill = TRUE, int xor = FALSE,
        const UI_REGION *clipRegion = ZIL_NULLP(UI_REGION),
        ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextHeight(const ZIL_ICHAR *string,
        ZIL_SCREENID screenID = ID_SCREEN,
        ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int TextWidth(const ZIL_ICHAR *string,
        ZIL_SCREENID screenID = ID_SCREEN,
        ZIL_LOGICAL_FONT font = FNT_DIALOG_FONT);
virtual int VirtualGet(ZIL_SCREENID screenID, int left, int top,
        int right, int bottom);
virtual int VirtualPut(ZIL_SCREENID screenID);
};

```

## General Members

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

- *XFONT* is a structure that contains the following font information:

*fontStruct* is a pointer to the X font structure, **XFontStruct**.

*fontList* is a list of X Window fonts created from *fontStruct*.

*fontSet* is used in Unicode mode only. It is a set of all fonts required to display characters for a given locale.

- *fontTable* is an array of font handles for X Windows. The following entries are pre-defined by OpenZinc:

**FNT\_SMALL\_FONT**—A small font similar in size to a font that might be used to display an icon's text string. The operating system's window manager is responsible for displaying the text on an icon, so this font is not typically used by OpenZinc.

**FNT\_DIALOG\_FONT**—A font that is used when text is displayed on window objects (e.g., UIW.BUTTON, UIW\_STRING, UIW\_TEXT, etc.)

**FNT\_SYSTEM\_FONT**—A slightly larger font similar in size to a font that might be used to display a window's title. The operating system's window manager is responsible for displaying the title of a window, so this font is not typically used by OpenZinc.

See the description of the *UI\_WINDOW\_OBJECT::font* member variable in "Chapter 43—UI\_WINDOW\_OBJECT" for information on specifying which font an object uses.

**NOTE:** All member functions use the standard OpenZinc screen pixel coordinates with (0,0) being the top-left corner of the display. This is done to remain consistent across platforms.

## **UI\_XT\_DISPLAY::UI\_XT\_DISPLAY**

### **Syntax**

```
#include <ui_dsp.hpp>
```

```
UI_XT_DISPLAY(int *argc = NULL, char **argv = NULL,  
             char *appClass = "ZincApp",  
             XrmOptionDescList options = ZIL_NULLP(XrmOptionDescRec),  
             Cardinal numOptions = 0, String *fallbackResources = ZIL_NULLP(String));
```

## 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 `UI_XT_DISPLAY` class object. All parameters for this constructor are passed to `XtAppInitialize()`. For more details on the use of these parameters, see the description of `XtAppInitialize()` in the *X Toolkit Intrinsic Reference Manual*.

- `argcin` is a pointer to an integer containing the number of arguments passed to function `main()`. `argc` points to a `1` if the program was invoked with no command-line arguments. If the program was invoked with one command-line argument, `argc` will point to a `2`, etc. `argc` is used to determine the number of parameters contained in `argv`.
- `argvin` is a pointer to an array of character strings that contain the actual command-line parameters. For example, if the program `TEST` were invoked with a `/C` switch, `argv[0]` would point to `"TEST"` and `argv[1]` would point to `/C"`.
- `appClassin` is a pointer to a character string denoting the class name of the application being executed. This identifies the name of the resource file used to initialize resources.
- `optionsin` describes how to parse the command line.
- `numOptionsin` specifies how many options were provided in `options`.
- `fallbackResourcesin` is a list of resource file entries that is used if the resource file specified by `appClass` cannot be found.

## Example

```
#include <ui_win.hpp>

int main(int argc, char **argv)
{
    // Initialize the display.
    UI_DISPLAY *display = new UI_XT_DISPLAY(&argc, argv, "ZincApp");

    // Initialize the event manager.
```

```

    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);

    *eventManager
      + new UID_KEYBOARD
      + new UID_MOUSE
      + new UID_CURSOR;

    // Initialize the window manager.
    UI_WINDOW_MANAGER *windowManager =
      new UI_WINDOW_MANAGER(display, eventManager);

    // Clean up.
    delete windowManager;
    delete eventManager;
    delete display;
    return (0);
}

```

## UI\_XT\_DISPLAY::~~UI\_XT\_DISPLAY

### Syntax

```

#include <ui_dsp.hpp>

~UI_XT_DISPLAY(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 UI\_XT\_DISPLAY class. Care should be taken to only destroy a UI\_XT\_DISPLAY class that is not attached to another associated object.



## CHAPTER 45 - UID CURSOR

The `UID_CURSOR` class is used to display a blinking cursor on the screen. It is used by objects that can be edited in order to show the end-user's position within the field. In text mode, this class uses BIOS calls to enable or disable the blinking hardware cursor. In DOS graphics mode, this class paints a blinking cursor on the screen. In environments other than DOS and Curses, the operating system handles the management of the cursor (in some operating systems, the edit cursor is referred to as a "caret"). In those environments, the `UID_CURSOR` class provides little, if any, control over the cursor.

The `UID_CURSOR` class is declared in `UI_EVT.HPP`. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UID_CURSOR : public UI_DEVICE,
    public ZIL_INTERNATIONAL
{
public:
    static int blinkRate;

    DEVICE_IMAGE image;
    UI_POSITION position;

    UID_CURSOR(ZIL_DEVICE_STATE State = D_OFF,
        DEVICE_IMAGE image = DC_INSERT);
    virtual ~UID_CURSOR(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);

protected:
    UI_POSITION offset;

    virtual void Poll(void);
};
```

### General Members

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

- *blinkRate* identifies the rate at which the cursor will blink. This value is in hundredths of seconds. This member is used for DOS and Curses only.
- *image* identifies the type of cursor being displayed on the screen. Its value may either be `DC_INSERT` or `DC_OVERSTRIKE`. In DOS graphics mode, if *image* is `DC_INSERT`, the cursor device displays a thick vertical bar cursor on the screen. If it is `DC_OVERSTRIKE`, the cursor device displays a thin vertical bar cursor on the screen. In DOS text mode, the `DC_INSERT` cursor is a wide box and the `DC_OVERSTRIKE` cursor is a short, wide underline. This member is used for DOS only.

- *position* contains the cursor's true screen position (based on the screen's 0,0 left-top based coordinates). The value of this structure depends on the type of display mode in which the application is running. For example, a cursor positioned in the middle of the screen may contain a *position.column* value of 40 and *position.line* value of 12, if the application is running in text mode. The same cursor position, however, may produce values of 320 and 240 if the application is running in graphics mode. This member is used for DOS and Curses only.
- *offset* is an offset, from *position*, where the cursor image will be displayed. This member is used for DOS and Curses only.

## UID\_CURSOR::UID CURSOR

### Syntax

```
#include <ui_evt.hpp>
```

```
UID_CURSOR(ZIL_DEVICE_STATE state = D_OFF,  
           DEVICE_IMAGE image = DC_INSERT);
```

### 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 UID\_CURSOR class object. It should be called after the display and Event Manager constructors have been called.

- *state<sub>in</sub>* is the initial state of the cursor device. The cursor device may be initialized to one of the following states (defined in **UI\_EVT.HPP**):

**D\_HIDE**—Initializes the cursor to be on but not visible. If the cursor is blinking, its state is either D\_ON or D\_HIDE. Thus, the cursor will only be invisible until it blinks back on. The blink rate is controlled by the *blinkRate* member. The initial state mask is DC\_INSERT.

D\_OFF—Initializes the cursor to be off. In this state, the cursor is not shown on the screen. This is the default value if no argument is provided.

D\_ON—Initializes the cursor to be on and visible. The initial state mask is DCJNSERT.

- *image<sub>m</sub>* identifies the initial type of cursor being displayed on the screen. Its value may be one of the following types (defined in UI\_EVT.HPP):

DC\_INSERT—The cursor device displays a thick vertical bar cursor in graphics mode, or a wide block in text mode. This is the default value if no argument is provided.

DC\_OVERSTRIKE—The cursor device displays a thin vertical bar cursor in graphics mode, or a short, wide underline in text mode.

## Example

```
#include <ui_evt.hpp>

main()
{
    // Attach the keyboard, mouse and cursor devices.
    UI_TEXT_DISPLAY *display = new UI_TEXT_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
      + new UID_KEYBOARD
      + new UID_MOUSE
      + new UID_CURSOR;

    // This automatically calls the keyboard, mouse and cursor destructors,
    delete eventManager;
    delete display;
    return (0);
}
```

## UID\_CURSOR::~~UID\_CURSOR

### Syntax

```
#include <ui_evt.hpp>

virtual ~UID_CURSOR(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 `UID_CURSOR` object. Care should be taken to only destroy a cursor device that is not attached to the Event Manager.

## Example

```
#include <ui_evt.hpp>

ExampleFunction()
{
    // Attach the keyboard, mouse and cursor devices.
    UI_TEXT_DISPLAY *display = new UI_TEXT_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;

    // This automatically calls the keyboard, mouse and cursor dest
    delete eventManager;
    delete display;
    return (0);
}
```

## UID\_CURSOR::Event

### Syntax

```
#include <ui_evt.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 messages sent to the cursor device. It is declared virtual so that any derived cursor class can override its default operation.

- *returnValue<sub>out</sub>* is the current state of the cursor device. This value will be `D_OFF`, `DC_INSERT` or `DC_OVERSTRIKE`.
- *event<sub>in</sub>* contains a run-time message for the cursor object. The following messages (declared in `UI_EVT.HPP`) are processed by the **Event()** function:

`D_HIDE`—Blinks the cursor off. If the cursor is blinking, its state is `D_ON` when it is visible and `D_HIDE` when it is not visible. Thus, no cursor will be visible until it blinks back on. The blink rate is controlled by the *blinkRate* member.

`D_OFF`—Turns off the cursor. If the cursor is off, no cursor is shown on the screen.

`D_ON`—Turns on the cursor. If the cursor is on, a cursor is shown on the screen until it is time to blink off.

`D_STATE`—Returns the current state of the cursor. If the cursor is on, the *image* will be returned.

`DC_INSERT`—Turns the cursor on and enables the insert cursor.

`DC_OVERSTRIKE`—Turns the cursor on and enables the overstrike cursor.

`S_DEINITIALIZE`—De-initializes the cursor device.

`S_INITIALIZE`—Initializes the cursor device.

`S_POSITION`—Changes the screen position of the cursor. If this message is sent, *event.position.column* and *event.position.line* must contain the run-time

display position of the cursor on the screen. The values of *event.position.column* and *event.position.line* depend on the type of display mode in which the application is running. For example, if the cursor is to be positioned at the center of the screen while the application is running in text mode (e.g., an 80 column by 25 line screen) the position values should be:

```
event.position.column = 40;
event.position.line = 13;
```

If, on the other hand, the application is running in a 640 column by 480 line graphics mode, the position values should be:

```
event.position.column = 320;
event.position.line = 240;
```

If the cursor is in a D\_OFF state, the position change will be reflected when the cursor is turned back on.

The state of the cursor device may also be changed using the **UI\_EVENT\_MANAGER::Event()** or **UI\_EVENT\_MANAGER::DeviceState()** functions.

**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.

## Example

```
#include <ui_evt.hpp>

ExampleFunction()
{
    // Attach the keyboard to the event manager.
    UI_DISPLAY *display = new UI_TEXT_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    UID_CURSOR *cursor = new UID_CURSOR;
    *eventManager + cursor;

    // Change the cursor to insert mode.
    UI_EVENT event;
    event.type = DC_INSERT;
    cursor->Event(event);

    // Reposition the cursor the top-left side of the screen,
    event.type = S_POSITION;
    event.position.column = event.position.line = 0;
    cursor->Event(event);
}
```

```
}
```

## UID\_CURSOR::Poll

### Syntax

```
#include <ui_evt.hpp>

virtual void Poll(void);
```

### 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 responsible for blinking the cursor on and off if the cursor is on and the application is running in DOS graphics mode. In text mode, this function has no effect. In addition, if the cursor is turned off, this function has no effect. This function is declared virtual so that any derived cursor class can override its default operation.

### Example

An example of the **Poll()** member function is presented in **UI\_DEVICE::Poll()**.



## CHAPTER 46 - UID\_KEYBOARD

The `UID_KEYBOARD` class is used to manage the keyboard device. This class handles events generated by the hardware keyboard device. Most compiler libraries have a set of functions to get input from the keyboard (e.g., `getch()`, `getchar()`). However, because OpenZinc Application Framework is an event-driven system, the keyboard is interfaced with other devices, such as a mouse, to provide smooth control of the user's input. In environments other than DOS and Curses, the operating system handles the management of the keyboard. In those environments, the `UID_KEYBOARD` class provides little, if any, control over the keyboard.

The `UID_KEYBOARD` class is declared in `UI_EVT.HPP`. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UID_KEYBOARD : public UI_DEVICE,
    public ZIL_INTERNATIONAL
{
public:
    static EVENT_TYPE breakHandlerSet;

    UID_KEYBOARD(ZIL_DEVICE_STATE State = D_ON);
    virtual ~UID_KEYBOARD(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);

protected:
    virtual void Poll(void);
};
```

### General Members

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

- *breakHandlerSet* defines what action is taken if the end-user attempts to break out of the program by hitting <Ctrl+Break> or <Ctrl+C>. If one of these key combinations is hit, the value of *breakHandlerSet* is placed on the event queue as an event. By default, *breakHandlerSet* is `L_EXIT`, which would cause the program to exit. If *breakHandlerSet* is set to `L_EXIT_FUNCTION`, the exit function associated with the Window Manager will be called, perhaps allowing the end-user to confirm that the application should be closed. The example below shows how this can be done:

```
UI_APPLICATION::Main()

    // Reset the break handler.
    UID_KEYBOARD::breakHandlerSet = L_EXIT_FUNCTION;
```

```
}
```

## Keyboard event information

The keyboard device provides the following event information (declared in `UI_EVT.HPP`) when a keyboard event is placed on the event queue:

```
struct ZIL_EXPORT_CLASS UI_KEY
{
    ZIL_RAW_CODE shiftState;
    ZIL_RAW_CODE value;
};

struct ZIL_EXPORT_CLASS UI_EVENT
{
    EVENT_TYPE type;           // The type of event (E_KEY).
    ZIL_RAW_CODE rawCode;     // The key's raw scan code.
    ZIL_RAW_CODE modifiers;

    union
    {
        UI_KEY key;           // The key information.
        UI_REGION region;
        UI_POSITION position;
        UI_SCROLL_INFORMATION scroll;
        void *data;
    };

    // Member functions are described in the UI_EVENT reference chapter.
};
```

- *type* is the event type. The `UID_KEYBOARD` device always generates an `E_KEY` event.
- *rawCode* is the key's raw scan code. `UI_MAP.HPP` contains **const** values for raw scan codes. Here are a few values for DOS:

```
const ZIL_RAW_CODE ENTER           = 0x1COD
const ZIL_RAW_CODE ESCAPE          = 0x011B
const ZIL_RAW_CODE F1               = 0x3B00
const ZIL_RAW_CODE GRAY_UP_ARROW   = 0x48E0
```

- *modifiers* is a flag field that indicates the shift state of the keyboard.
- *key.shiftState* is the shift state of the keyboard. The shift state may contain one or more of the following flags (declared in `UI_EVT.HPP`):

`S_ALT`—Indicates that the <Alt> key was pressed.

**S\_CAPS\_LOCK**—Indicates that the <Caps-Lock> key was on.

**S\_CTRL**—Indicates that the <Ctrl> key was pressed.

**S\_INSERT**—Indicates that the <Ins> key was on.

**S\_LEFT\_SHIFT**—Indicates that the <Left-Shift> key was pressed.

**S\_NUM\_LOCK**—Indicates that the <Num-Lock> key was on.

**S\_RIGHT\_SHIFT**—Indicates that the <Right-Shift> key was pressed.

**S\_SCROLL\_LOCK**—Indicates that the <Scroll-Lock> key was on.

- *key.value* is the key's value. It is not the scan code.

## **UID\_KEYBOARD::UID\_KEYBOARD**

---

### **Syntax**

```
#include <ui_evt.hpp>
```

```
UID_KEYBOARD(ZIL_DEVICE_STATE state = D_ON);
```

### **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 `UID_KEYBOARD` class object.

- *state<sub>in</sub>* is the initial state of the keyboard device. The keyboard device may be set to one of the following states (declared in `UI_EVT.HPP`):



**D\_OFF**—Turns the keyboard device off. If the keyboard state is set to **D\_OFF**, events are removed from the keyboard buffer but are not placed in the event queue (i.e., they are discarded).

**D\_ON**—Turns the keyboard device on. If the keyboard is on, keyboard events will be placed on the event queue. This is the default value if no argument is provided.

The state of the **UID\_KEYBOARD** device can be changed at run-time using the **UID\_KEYBOARD::Event()**, **UI\_EVENT\_MANAGER::Event()** or **UI\_EVENT\_MANAGER::DeviceState()** function calls.

## Example

```
#include <ui_evt.hpp>

main()
{
    // Attach the keyboard, mouse and cursor devices.
    UI_TEXT_DISPLAY *display = new UI_TEXT_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;

    // This automatically calls the keyboard, mouse and cursor destructors,
    delete eventManager;
    delete display;
    return (0);
}
```

## UID\_KEYBOARD::~~UID\_KEYBOARD

### Syntax

```
#include <ui_evt.hpp>

virtual ~UID_KEYBOARD(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 UID\_KEYBOARD object. Care should be taken to only destroy a keyboard device that is not attached to the Event Manager.

## Example

```
#include <ui_evt.hpp>

main()
{
    // Attach the keyboard, mouse and cursor devices.
    UI_DISPLAY *display = new UI_MSC_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;

    // This automatically calls the keyboard, mouse and cursor destructors,
    delete eventManager;
    delete display;
}
```

## UID\_KEYBOARD::Event

### Syntax

```
#include <ui_evt.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 messages sent to the keyboard device. It is declared virtual so that any derived keyboard class can override its default operation.

- *returnValue<sub>out</sub>* is the current state of the keyboard device. This value will be D\_OFF or D\_ON.
- *event<sub>in</sub>* contains a run-time message for the keyboard object. The following events are processed by **Event()**:

**D\_OFF**—Turns the keyboard device off. If the keyboard state is set to D\_OFF, events are removed from the keyboard buffer but are not placed in the event queue (i.e., they are discarded).

**D\_ON**—Turns the keyboard device on. If the keyboard is on, keyboard events will be placed on the event queue. This is the default value if no argument is provided.

**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.

## Example

```
#include <ui_evt.hpp>

main()
{
    // Attach the keyboard to the event manager.
    UI_DISPLAY *display = new UI_MSC_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    UID_KEYBOARD *keyboard = new UID_KEYBOARD;
    *eventManager + keyboard;

    // Turn the keyboard off directly.
```

```
UI_EVENT event;  
event.type = D_OFF;  
keyboard->Event(event) ?  
  
}
```

## UID\_KEYBOARD::Poll

### Syntax

```
#include <ui_evt.hpp>  
  
virtual void Poll(void);
```

### 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 places any pending keyboard events on the event queue.

### Example

An example of the **Poll()** member function is given in **UI\_DEVICE::Poll()**.



## CHAPTER 47 - UID\_MOUSE

The UID\_MOUSE class is used to manage the mouse device. This class handles events generated by the hardware mouse device and controls the presentation of the mouse image. In environments other than DOS and Curses, the operating system handles the management of the mouse. In those environments, the UID\_MOUSE class provides little, if any, control over the mouse.

The UID\_MOUSE class is declared in **UI\_EVT.HPP**. Its public and protected members are:

```
class ZIL_EXPORT_CLASS UID_MOUSE : public UI_DEVICE
{
public:
#if defined(ZIL_MSDOS) && !defined(ZIL_TEXT_ONLY)
    static int defaultInitialized;
    static ZIL_ICHAR _className[];
#endif
    DEVICE_IMAGE image;
    UI_POSITION position;

    UID_MOUSE(ZIL_DEVICE_STATE State = D_ON, DEVICE_IMAGE image = DM_WAIT);
    virtual ~UID_MOUSE(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);

protected:
    UI_POSITION offset;

    virtual void Poll(void);
    // I18N member variables and functions.
#if defined(ZIL_MSDOS) && !defined(ZIL_TEXT_ONLY)
    const ZIL_DECORATION *myDecorations;
#endif
public:
#if defined(ZIL_MSDOS)
    void MouseMove(int deltaX, int deltaY);
#endif
};
```

### General Members

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

- *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.

- *\_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 `UID_MOUSE` class, *\_className* is "UID\_MOUSE."
- *image* identifies the type of mouse cursor being displayed on the screen, *image* may be one of the following:

`DM_DIAGONAL_ULLR`—Displays the image shown when sizing the top-left or bottom-right corner of a window.

`DM_DIAGONAL_LLUR`—Displays the image shown when sizing the top-right or bottom-left corner of a window.

`DM_DRAG_COPY`—Displays the image shown when dragging an object to copy it.

`DM_DRAG_MOVE`—Displays the image shown when dragging an object to move it.

`DM_EDIT`—Displays the image shown when positioned over an editable field.

`DM_HORIZONTAL`—Displays the image shown when sizing a window horizontally.

`DM_MOVE`—Displays the image shown when indicating that the object is to be moved.

`DM_POSITION_IMAGE`—Displays the image shown when indicating that something is to be positioned by the device.

`DM_VERTICAL`—Displays the image shown when sizing a window vertically.

`DM_VIEW`—Displays the default image, typically an arrow.

`DM_WAIT`—Displays the image shown to indicate to the user that some processing is taking place and that he should wait.

NOTE: Because OpenZinc allows the graphical operating systems to handle their images, not all of these images may be supported in all environments.

- *position* contains the mouse cursor's true screen position (based on the screen's 0,0 left-top based coordinates). The value of this structure depends on the type of display

mode in which the application is running. For example, a mouse cursor positioned in the middle of the screen may contain a *position.column* value of 40 and *position.line* value of 12, if the application is running in text mode. The same mouse cursor position, however, may produce values of 320 and 240 if the application is running in VGA graphics mode.

- *offset* is an offset, from *position*, where the mouse image will be displayed.

## Mouse event information

The mouse device provides the following event information (declared in UI\_EVT.HPP) when a mouse event is placed on the event queue:

```

Struct UI_POSITION
{
    int column;           // The mouse column position,
    int line;            // The mouse line position.
};

Struct ZIL_EXPORT_CLASS UI_EVENT
{
    EVENT_TYPE type;     // The type of event (E_MOUSE).
    ZIL_RAW_CODE rawCode; // The keyboard and mouse scan code
    ZIL_RAW_CODE modifiers; // Keyboard modifier key scan codes.

#if defined(ZIL_MSWINDOWS)
    MSG message;        // Windows message field.
#elif defined(ZIL_OS2)
    QMSG message;      // OS/2 message field.
#elif defined(ZIL_MOTIF)
    XEvent message;    // Motif message field.
#endif

    UI_KEY key;
    UI_REGION region;
    UI_POSITION position; // The mouse position
    UI_SCROLL_INFORMATION scroll;

    void *data;
};

// Member functions are described in the UI_EVENT reference chapter.

```

- *type* is the event type. The mouse device always generates an E\_MOUSE event in DOS and Curses.



- *rawCode* contains the keyboard's shift state and the mouse's button state. The possible state values are as follows:

**M\_LEFT**—The left mouse button is pressed. The Macintosh uses this value with single-button mice.

**M\_LEFT\_CHANGE**—The left mouse button state has changed. If the **M\_LEFT\_CHANGE** and **M\_LEFT** flags are set, the left button has just been pressed. Otherwise, the left button has just been released. The Macintosh uses this value with single-button mice.

**M\_MIDDLE**—The middle mouse button is pressed. This flag will only be set when a three-button mouse is in use.

**M\_MIDDLE\_CHANGE**—The middle mouse button state has changed. If the **M\_MIDDLE\_CHANGE** and **M\_MIDDLE** flags are set, the middle button has just been pressed. Otherwise, the middle button has just been released. This flag will only be set when a three-button mouse is in use.

**M\_RIGHT**—The right mouse button is pressed.

**M\_RIGHT\_CHANGE**—The right mouse button state has changed. If the **M\_RIGHT\_CHANGE** and **M\_RIGHT** flags are set, the right button has just been pressed. Otherwise, the right button has just been released.

**S\_ALT**—Indicates that the <Alt> key was pressed.

**S\_CAPS\_LOCK**—Indicates that the <Caps-Lock> key was on.

**S\_CTRL**—Indicates that the <Ctrl> key was pressed.

**S\_INSERT**—Indicates that the <Ins> key was on.

**S\_LEFT\_SHIFT**—Indicates that the <Left-Shift> key was pressed.

**S\_NUM\_LOCK**—Indicates that the <Num-Lock> key was on.

**S\_RIGHT\_SHIFT**—Indicates that the <Right-Shift> key was pressed.

**S\_SCROLL\_LOCK**—Indicates that the <Scroll-Lock> key was on.

**NOTE:** The `M_TOP_CHANGE` and `M_BOTTOM_CHANGE` values are only used when a window object is to be sized. They are not set by the `UID_MOUSE` class.

- *modifiers* is a flag field that indicates the shift state of the keyboard.
- *position.column* is the mouse's horizontal position. In graphics mode, this value is given in pixel coordinates. In text mode, this value is given in character coordinates.
- *position.line* is the mouse's vertical position. In graphics mode, this value is given in pixel coordinates. In text mode, this value is given in character coordinates.

## **UID\_MOUSE::UID\_MOUSE**

### **Syntax**

```
#include <ui_evt.hpp>
```

```
UID_MOUSE(ZIL_DEVICE_STATE state = D_ON,  
          DEVICEIMAGE image = DM_WAIT);
```

### **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 `UID_MOUSE` class object.

- *state<sub>in</sub>* is the initial state of the mouse device. The mouse device may be set to one of the following states (declared in `UI_EVT.HPP`):

**D\_HIDE**—Initializes the mouse to be on but not visible.

**D\_OFF**—Turns the mouse device off. If the mouse is off, no mouse events will be placed on the event queue.

**D\_ON**—Turns the mouse device on. If the mouse is on, mouse events will be placed on the event queue. This is the default value if no argument is provided.

The state of the `UID_MOUSE` device can be changed at run-time using the `UID_MOUSE::Event()`, `UI_EVENT_MANAGER::Event()` or `UI_EVENT_MANAGER::DeviceState()` function calls.

- `imagein` identifies the initial mouse image to be displayed. See the description of the `image` member above for more details.

## Example

```
#include <ui_evt.hpp>

main()
{
    // Attach the keyboard, mouse and cursor devices.
    UI_DISPLAY *display = new UI_MSC_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
        + new UID_KEYBOARD
        + new UID_MOUSE
        + new UID_CURSOR;

    // This automatically calls the keyboard, mouse and cursor destructors,
    delete eventManager;
    delete display;
    return (0);
}
```

## UID\_MOUSE::~~UID\_MOUSE

### Syntax

```
#include <ui_evt.hpp>

virtual ~UID_MOUSE(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 UID\_MOUSE object. Care should be taken to only destroy a mouse device that is not attached to the Event Manager.

## Example

```
#include <ui_evt.hpp>

main()
{
    // Attach the keyboard, mouse and cursor devices.
    UI_DISPLAY *display = new UI_MSC_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
    *eventManager
      + new UID_KEYBOARD
      + new UID_MOUSE
      + new UID_CURSOR;

    // This automatically calls the device destructors,
    delete eventManager;
    delete display;
    return (0);
}
```

## UID\_MOUSE::Event

### Syntax

```
#include <ui_evt.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 messages sent to the mouse device. It is declared

virtual so that any derived mouse class can override its default operation.

- *returnValue<sub>out</sub>* is the current state of the mouse device. This value will be D\_OFF, D\_ON or D\_HIDE.
- *event<sub>in</sub>* contains a run-time message for the mouse device. The following events are processed by Event():

DM\_DIAGONAL\_ULLR—Displays the image shown when sizing the top-left or bottom-right corner of a window.

DM\_DIAGONAL\_LLUR—Displays the image shown when sizing the top-right or bottom-left corner of a window.

DM\_DRAG\_COPY—Displays the image shown when dragging an object to copy it.

DM\_DRAG\_MOVE—Displays the image shown when dragging an object to move it.

DM\_EDIT—Displays the image shown when positioned over an editable field.

DM\_HORIZONTAL—Displays the image shown when sizing a window horizontally.

DM\_MOVE—Displays the image shown when indicating that the object is to be moved.

DM\_POSITION\_IMAGE —Displays the image shown when indicating that something is to be positioned by the device.

DM\_VERTICAL—Displays the image shown when sizing a window vertically.

DM\_VIEW—Displays the default image, typically an arrow.

DM\_WAIT—Displays the image shown to indicate to the user that some processing is taking place and that he should wait.

D\_HIDE—Hides the mouse.

D\_OFF—Turns the mouse device off. If the mouse is off, no mouse events will be placed on the event queue.

D\_ON—Turns the mouse device on. If the mouse is on, mouse events will be placed on the event queue. This is the default value if no argument is provided.

D\_STATE—Returns the current state of the mouse. If the mouse is on, the *image* will be returned.

S\_DEINITIALIZE—De-initializes the device.

SJNITIALIZE—Initializes the device.

S\_POSITION—Changes the screen position of the mouse. If this message is sent, *event.position.column* and *event.position.line* must contain the run-time display position of the mouse on the screen. The values of *event.position.column* and *event.position.line* depend on the type of display mode in which the application is running. For example, if the mouse is to be positioned at the center of the screen while the application is running in text mode (e.g., an 80 column by 25 line screen) the position values should be:

```
event.position.column = 40;
event.position.line = 13;
```

If, on the other hand, the application is running in a 640 column by 480 line graphics mode, the position values should be:

```
event.position.column = 320;
event.position.line = 240;
```

If the mouse is in a D\_OFF state, the position change will be reflected when the mouse is turned back on.

The state of the mouse device may also be changed using the UI\_EVENT\_MANAGER::Event() or UI\_EVENT\_MANAGER::DeviceState() functions.

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.

## Example

```
#include <ui_evt.hpp>

main()
{
    UI_DISPLAY *display = new UI_GRAPHICS_DISPLAY;
    UI_EVENT_MANAGER *eventManager = new UI_EVENT_MANAGER(display);
}
```

```

UID_MOUSE *mouse = new UID_MOUSE;
*eventManager + mouse;

// Turn on the mouse wait cursor.
UI_EVENT event;
event.type = DM_WAIT;
mouse->Event(event);

// Turn on the mouse arrow cursor.
UI_EVENT event;
event.type = DM_VIEW;
mouse->Event(event);
}

```

## UID\_MOUSE::MouseMove

### Syntax

```

#include <ui_evt.hpp>

void MouseMove(int deltaX, int deltaY);

```

### 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 moves the mouse from its current position by the amount indicated in *deltaX* and *deltaY*. The mouse will not be allowed to move off the edge of the screen.

- *deltaX<sub>in</sub>* is the distance to move the mouse horizontally.
- *deltaY<sub>in</sub>* is the distance to move the mouse vertically.

## **UID\_MOUSE::Poll**

### **Syntax**

```
#include <ui_evt.hpp>

virtual void Poll(void);
```

### **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 places any pending mouse events on the event queue.

### **Example**

An example of the **Poll()** member function is given in **UI\_DEVICE::Poll()**.

## **Internationalization Members**

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

- *myDecorations* is the ZIL\_DECORATION object that contains the images for this object.





## CHAPTER 48 - UID TIMER

The `UID_TIMER` class is used to notify an object when a specified length of time has elapsed. When the timer expires, it sends a message to the object and, if desired, can also place a message on the event queue. The timer should not be used for time-critical tasks that require notification at exact time intervals. The `UID_TIMER` class is not an interrupt driven class and must wait for current processes to give up the CPU before it can process a timer expiration. It is possible, depending on the system load, for several timer intervals to elapse before the timer is able to generate a timer message. Thus, the object requesting timer services should look at the current time when it receives a timer notification if it needs to know exactly how much time has elapsed since the last notification.

A timer can be set to expire at a single interval. If more than one timer interval is required, simply create another timer with the new interval.

A timer device is attached to the Event Manager. Objects requesting timer services, if any, are attached to the timer by sending the timer an `S_ADD_OBJECT` message. When the object no longer requires timer services, it should notify the timer by sending an `S_SUBTRACT_OBJECT` message. It is very important that an object be removed from the timer device if it is being deleted. Otherwise, the timer may attempt to send a timer message to the object after its memory has been freed, resulting in undefined, and likely fatal, behavior. An object can remove itself from the timer either in its destructor or when it receives an `S_DEINITIALIZE` message.

The `UID_TIMER` class is declared in `UI_EVT.HPP`. Its public and protected members are:

```
class UID_TIMER : public UI_DEVICE
{
    #if defined(ZIL_MOTIF)
        friend void TimerCallback(XtPointer client_data, XtIntervalId *id);
    #endif

public:
    TMR_FLAGS tmrFlags;

    UID_TIMER(ZIL_DEVICE_STATE state = D_ON, unsigned interval = 0,
             TMR_FLAGS tmrFlags = TMR_NO_FLAGS);
    virtual ~UID_TIMER(void);
    virtual EVENT_TYPE Event(const UI_EVENT &event);

    // NOTIFY_ELEMENT and NOTIFY_LIST should be treated as protected,
    // but this causes compiler errors on some systems.
    class NOTIFY_ELEMENT : public UI_ELEMENT
    {
    public:
        UI_WINDOW_OBJECT *notifyObject;
        NOTIFY_ELEMENT(UI_WINDOW_OBJECT *object);
        virtual ~NOTIFY_ELEMENT(void);
        NOTIFY_ELEMENT *Next(void);
    };
};
```

```

        NOTIFY_ELEMENT *Previous(void);
    };

class NOTIFY_LIST : public UI_LIST
{
public:
    NOTIFY_ELEMENT *First(void);
    NOTIFY_ELEMENT *Last(void);
};

protected:
    unsigned msec;
    ZIL_UTIME interval;
    ZIL_UTIME lastTime;
    NOTIFY_LIST notifyList;

    virtual void Poll(void);

```

## General Members

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

- *tmrFlags* are flags that define the operation of the UID\_TIMER class. A full description of the timer flags is given in the UID\_TIMER constructor.
- *msec* is the length of time, in milliseconds, that must elapse before the timer generates a timer message.
- *interval* is the length of time, in milliseconds, that must elapse before the timer generates a timer message. This representation of the interval (i.e., as a ZIL\_UTIME) is used to ensure that time comparisons that cross the midnight boundary are calculated correctly.
- *lastTime* is the last time at which the timer device expired and sent a timer message. This time is used as a reference to determine if the time specified by *interval* has expired.
- *notifyList* is the list of objects that have requested timer services from this timer device. This list is a NOTIFYJLIST of NOTIFY\_ELEMENT objects. These classes will not be discussed, as they simply maintain a doubly-linked list of pointers.

## UID\_TIMER::UID\_TIMER

### Syntax

```
#include <ui_evt.hpp>
```

```
UID_TIMER(ZIL_DEVICE_STATE state = D_ON, unsigned interval = 0,  
          TMR_FLAGS tmrFlags = TMR_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 UID\_TIMER class object.

- *state<sub>in</sub>* is the initial state of the timer device. The timer device may be initialized to one of the following states (defined in **UI\_EVT.HPP**):

**D\_OFF**—Initializes the timer to be off. In this state, the timer will not generate timer messages.

**D\_ON**—Initializes the timer to be on. In this state, the timer will generate timer messages when the timer expires.

- *interval<sub>in</sub>* is the length of time, in milliseconds, that must elapse before the timer generates a timer message. The resolution of the timer (i.e., how short the intervals can be) depends largely on the operating system and the system load. Because timer messages are generated from the **Poll()** function, how often the **Poll()** function is called can affect the resolution. For example, the OS/2 system timer can "fire" 18.2 times a second (the hardware clock tick rate). The OS/2 timer is only used to notify the application in case the application has become idle (in which case the **Poll()** is not being called). So, we are "assured" (with the limitations mentioned previously) of the timer activating at least 18.2 times each second in OS/2. Setting the Q\_NO\_BLOCK flag on calls to **UI\_EVENT\_MANAGER::Get()** in the main event loop and setting an interval of 0 will result in timer messages being generated as quickly as possible.

The maximum *interval* allowed varies depending on the operating system. Windows has the shortest maximum interval at 65535 milliseconds. DOS can have intervals of up to  $2^{32}$  milliseconds.

If *interval* is 0, the default, timer messages will be generated as quickly as possible. How often zero-interval timer messages are generated depends on the operating system, whether there are other processes running and how long the application takes to process a timer message.

- *tmrFlags<sub>in</sub>* are flags that define the operation of the UID\_TIMER class. The following flags (declared in **UI\_WIN.HPP**) control the general presentation of a UID\_TIMER class object:

**TMR\_NO\_FLAGS**—Does not associate any special flags with the UID\_TIMER class object.

**TMR\_QUEUE\_EVENTS**—Causes the timer to place a timer event on the event queue when the timer expires. Generally, this flag should only be set if no objects are requesting timer services directly, but it can be used if objects are requesting services.

## UID\_TIMER::~~UID\_TIMER

### Syntax

```
#include <ui_evt.hpp>

virtual ~UID_TIMER(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 UID\_TIMER object.

## UID\_TIMER::Event

### Syntax

```
#include <ui_evt.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 messages sent to the timer device. It is declared virtual so that any derived timer class can override its default operation.

- *returnValue<sub>out</sub>* is the current state of the timer device. This value will be D\_OFF or D\_ON.
- *event<sub>in</sub>* contains a run-time message for the timer object. The following messages (declared in **UI\_EVT.HPP**) are processed by the **Event( )** function:

**D\_OFF**—Turns off the timer. If the timer is off, it will not generate timer messages.

**D\_ON**—Turns on the timer. If the timer is on, it will generate timer messages when the timer expires.

**S\_DEINITIALIZE**—De-initializes the timer device. If necessary, the timer will turn off any timer processing that was previously set up with the operating system.

**S\_INITIALIZE**—Initializes the timer device. If necessary, the timer device will set up timer processing with the operating system.

S\_ADD\_OBJECT—Adds an object to the timer's *notifyList*. If this message is sent, *event.windowObject* must be a pointer to the object requesting timer services.

S\_SUBTRACT\_OBJECT—Removes an object from the timer's *notifyList*. If this message is sent, *event.windowObject* must be a pointer to the object requesting that it no longer receive timer services.

## UIP\_TIMER::Poll

### Syntax

```
#include <ui_evt.hpp>

virtual void Poll(void);
```

### 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 checks to see if the timer has expired. If so, it sends an E\_TIMER message to all objects that have requested timer services from this timer. It also places an E\_TIMER event on the event queue if the TMR\_QUEUE\_EVENTS flag is set for the timer. In both cases *event.device* will point to the timer device generating the message.

### Example

An example of the **Poll()** member function is presented in **UI\_DEVICE::Poll()**.

## CHAPTER 49 - ZIL\_BIGNUM

The ZIL\_IBIGNUM class is a lower-level class used to store and manipulate numerical values. It is not a window object. See "Chapter 1—UIW\_BIGNUM" of *Programmer's Reference Volume 2* for information about the bignum window object. The values handled by ZIL\_BIGNUM include both integer and real bignums with a default maximum of 30 digits to the left and 8 digits to the right of the decimal point.

The ZIL\_BIGNUM class is declared in **UI\_GEN.HPP**. Its public and protected members are:

```
#define NUMBER_WHOLE 30
#define NUMBER_DECIMAL 8
#if ZIL_WORD_SIZE > 16
#   define ZIL_DIGITS 8
#else
#   define ZIL_DIGITS 4
#endif

typedef ZIL_INT32 ZIL_IBIGNUM;
typedef double ZIL_RBIGNUM;
#if ZIL_DIGITS == 4
    typedef ZIL_UINT16 ZIL_NUMBER;
#elif ZIL_DIGITS == 8
    typedef ZIL_UINT32 ZIL_NUMBER;
#endif

class ZIL_EXPORT_CLASS ZIL_BIGNUM : public ZIL_INTERNATIONAL
{
    friend class ZIL_EXPORT_CLASS UIW_BIGNUM;

    friend ZIL_BIGNUM &abs(const ZIL_BIGNUM &number);
    friend ZIL_BIGNUM &ceil(const ZIL_BIGNUM &number);
    friend ZIL_BIGNUM &floor(const ZIL_BIGNUM &number);
    friend ZIL_BIGNUM &round(const ZIL_BIGNUM &number, int places = 0);
    friend ZIL_BIGNUM &truncate(const ZIL_BIGNUM &number, int places = 0);

public:
    ZIL_BIGNUM(void);
    ZIL_BIGNUM(ZIL_IBIGNUM value);
    ZIL_BIGNUM(ZIL_RBIGNUM value);
    ZIL_BIGNUM(const ZIL_ICHAR *string,
               const ZIL_ICHAR *decimalString = ZIL_NULLP(ZIL_ICHAR),
               const ZIL_ICHAR *signString = ZIL_NULLP(ZIL_ICHAR));
    ZIL_BIGNUM(const ZIL_BIGNUM &number);
    virtual ~ZIL_BIGNUM(void);
    void Export(ZIL_IBIGNUM *value);
    void Export(ZIL_RBIGNUM *value);
    void Export(ZIL_ICHAR *string, NMF_FLAGS nmFlags,
               const ZIL_ICHAR *decimalString = ZIL_NULLP(ZIL_ICHAR),
               const ZIL_ICHAR *signStr = ZIL_NULLP(ZIL_ICHAR));
    NMI_RESULT Import(ZIL_IBIGNUM value);
    NMI_RESULT Import(ZIL_RBIGNUM value);
    NMI_RESULT Import(const ZIL_BIGNUM &number);
    NMI_RESULT Import(const ZIL_ICHAR *string,
                     const ZIL_ICHAR *decimalString = ZIL_NULLP(ZIL_ICHAR),
                     const ZIL_ICHAR *signString = ZIL_NULLP(ZIL_ICHAR));

    ZIL_BIGNUM &operator=(const ZIL_BIGNUM &number);
    ZIL_BIGNUM &operator+(const ZIL_BIGNUM &number);
};
```



```

ZIL_BIGNUM &operator-(const ZIL_BIGNUM &number);
ZIL_BIGNUM ^operator*(const ZIL_BIGNUM &number);
ZIL_BIGNUM &operator++(void);
ZIL_BIGNUM &operator--(void);
ZIL_BIGNUM &operator+=(const ZIL_BIGNUM &number);
ZIL_BIGNUM &operator-=(const ZIL_BIGNUM &number);
int operator==(const ZIL_BIGNUM &number);
int operator!=(const ZIL_BIGNUM &number);
int operator>(const ZIL_BIGNUM &number);
int operator>=(const ZIL_BIGNUM &number);
int operator<(const ZIL_BIGNUM &number);
int operator<=(const ZIL_BIGNUM &number);
void SetLocale(const ZIL_ICHAR *localeName)

protected:
    const ZIL_LOCALE *myLocale;

```

## General Members

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

- *NUMBER\_WHOLE* is the number of digits allowed to the left of the decimal place. The default is to allow 30 digits to the left of the decimal place. To use numbers with more than 30 digits to the left of the decimal place, simply change the value of *NUMBER\_WHOLE* to the desired amount and recompile the bignum module. No other changes are necessary.
- *NUMBER\_DECIMAL* is the number of digits allowed to the right of the decimal place. The default is to allow 8 digits to the right of the decimal place. To use numbers with greater precision than 8 decimal places, simply change the value of *NUMBER\_DECIMAL* to the desired amount and recompile the bignum module. No other changes are necessary.
- *ZIL\_DIGITS* is used for number conversion and manipulation. The default value is 4, which allows the *ZIL\_BIGNUM* class to work with integer values of 32 bits or less. If this value is changed to 8, the *ZIL\_BIGNUM* class will work with integer values of 32 bits or more. The size of the integer used depends on the environment being compiled for.

NOTE: The *ZIL\_BIGNUM* class uses special number types to do numerical operations. With the *ZIL\_BIGNUM* class, use the following number types:

- *ZIL\_IBIGNUM* is an integral data type associated with *ZIL\_BIGNUM*. This type should be used when integer operations are done. It is defined to be of type

**ZIL\_INT32**, which will be a signed int of at least 32 bits, depending on the word size of the environment being compiled for.

- *ZIL\_RBIGNUM* is the real number data type associated with *ZIL\_BIGNUM*. This type should be used when real operations are done. It is defined to be of type **double**. Using this type will require that the floating point library be used which will increase, often significantly, the size of the executable. Unless an individual application requires that floating point numbers be used, it is recommended that the string equivalent of a decimal number be used instead of the floating point numbers (i.e., "1.1" vs. 1.1).
- *ZIL\_NUMBER* is a type used internally by the *ZIL\_IBIGNUM* class. Programmers need not use this type.
- *myLocale* is the *ZIL\_LOCALE* object that contains the formatting information for this object.

## ZIL\_BIGNUM::ZIL\_BIGNUM

### Syntax

```
#include <ui_gen.hpp>

ZIL_BIGNUM(void);
    or
ZIL_BIGNUM(ZIL_IBIGNUM value);
    or
ZIL_BIGNUM(ZIL_RBIGNUM value);
    or
ZIL_BIGNUM(const ZIL_ICHAR *string,
            const ZIL_ICHAR *decimalString = ZIL_NULLP(ZIL_ICHAR),
            const ZIL_ICHAR *signString = ZIL_NULLP(ZIL_ICHAR));
    or

ZIL_BIGNUM(const ZIL_BIGNUM &number);
```

### Portability

These functions are available on the following environments:

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

## Remarks

These overloaded constructors create a new `ZIL_BIGNUM` class object.

The first overloaded constructor creates a `ZIL_BIGNUM` object and initializes its value to zero.

The second overloaded constructor creates a `ZIL_BIGNUM` object and initializes its value with *value*.

- *value<sub>in</sub>* is a `ZIL_IBIGNUM` (integer) value to which the `ZIL_BIGNUM` object will be initialized.

The third overloaded constructor creates a `ZIL_BIGNUM` object and initializes its value with *value*.

- *value<sub>in</sub>* is a `ZIL_RBIGNUM` (real) value to which the `ZIL_BIGNUM` object will be initialized.

The fourth overloaded constructor creates a `ZIL_BIGNUM` object and initializes its value with *string*.

- *string<sub>m</sub>* is a character string that contains the value, either integral or real, to which the `ZIL_BIGNUM` object will be initialized.
- *decimalString<sub>m</sub>* is a pointer to the decimal character to be used in formatting the decimal number.
- *signString<sub>m</sub>* is a pointer to the sign character to be used in formatting the bignum.

The fifth overloaded constructor creates a `ZIL_BIGNUM` object and initializes its value with *number*.

- *number<sub>in</sub>* is another `ZIL_BIGNUM` object whose value will be copied into the `ZIL_BIGNUM` object being constructed.

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_IBIGNUM i = 4;
    ZIL_RBIGNUM r = 7.1;
}
```

```

ZIL_BIGNUM number1;
ZIL_BIGNUM *number2 = new ZIL_BIGNUM(i);
ZIL_BIGNUM *number3 = new ZIL_BIGNUM(r);
ZIL_BIGNUM *number4 = new ZIL_BIGNUM("100");
ZIL_BIGNUM *number5 = new ZIL_BIGNUM(&number1);

```

```

delete number5;
delete number4;
delete number3;
delete number2;

```

## ZIL\_BIGNUM::~~ZIL\_BIGNUM

### Syntax

```

#include <ui_gen.hpp>

virtual ~ZIL_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 ZIL\_BIGNUM object. Care should be taken to only destroy a ZIL\_BIGNUM class that is not attached to another associated object.

### Example

```

#include <ui_gen.hpp>
ExampleFunction()
{
    ZIL_BIGNUM *number = new ZIL_BIGNUM("100")

    delete number;
}

```

## ZIL\_BIGNUM::abs

### Syntax

```
friend ZIL_BIGNUM &abs(const ZIL_BIGNUM &number)\
```

### 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 a ZIL\_BIGNUM object containing the absolute value of the ZIL\_BIGNUM value passed in.

- *returnValue<sub>out</sub>* is a ZIL\_BIGNUM object containing the absolute value of *number*.
- *number<sub>in</sub>* is a ZIL\_BIGNUM object for which the absolute value is desired.

### Example

```
#include <ui_gen.hpp>

ExampleFunction( )
{
    ZIL_BIGNUM *firstValue;
    ZIL_BIGNUM secondValue("-100");
    ZIL_BIGNUM thirdValue("-100");

    firstValue = abs(secondValue + thirdValue);
}
```

## ZIL\_BIGNUM::ceil

### Syntax

```
friend ZIL_BIGNUM &ceil(const ZIL_BIGNUM &number)\
```

## 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 a ZIL\_BIGNUM object containing the ceiling value of the ZIL\_BIGNUM value. The ceiling of a bignum is considered to be the smallest integer that is greater than or equal to *number*.

- *returnValue<sub>out</sub>* is a ZIL\_BIGNUM object containing the ceiling value of *number*.
- *number<sub>in</sub>* is a ZIL\_BIGNUM object for which the ceiling value is desired.

## Example

```
#include <ui_gen.hpp>

ExampleFunction( )
{
    ZIL_BIGNUM *firstValue;
    ZIL_BIGNUM secondValue("100.6");

    firstValue = ceil(secondValue);
}
```

## ZIL\_BIGNUM::Export

### Syntax

```
#include <ui_gen.hpp>

void Export(ZIL_IBIGNUM *value);
    or
void Export(ZIL_RBIGNUM *value);
    or
void Export(ZIL_ICHAR *string, NMF_FLAGS nmFlags,
    const ZIL_ICHAR *decimalString = ZIL_NULLP(ZIL_ICHAR),
    const ZIL_ICHAR *signStr = 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

These overloaded functions return the numerical information through a function-specific numeric value.

The first two overloaded functions copy the bignum information into the *value* argument.

- *value<sub>in</sub>* is a numeric value. The following types are supported for *value*:

ZIL\_IBIGNUM—A type which will be a signed int of at least 32 bits, depending on the environment being compiled for. For an environment that uses 32-bit words or smaller, the value can be between -2,147,483,648 and 2,147,483,647 (32 bits, signed), inclusive. For environments that use 64-bit words, the value can be between -9,223,372,036,854,775,809 and 9,223,372,036,854,775,808 (64 bits, signed), inclusive.

ZIL\_RBIGNUM—A double precision floating point number.

The third overloaded function copies the number into the *string* argument. When this function is used, space must be previously allocated for *string* by the programmer.

- *string<sub>out</sub>* is a pointer to a string that represents the bignum's value.
- *nmFlags<sub>in</sub>* gives formatting information about the return bignum's value. The following flags (declared in UI\_GEN.HPP) are used to format the bignum string:

NMF\_COMMAS—Formats the bignum with commas (or the appropriate locale-specific thousands separator symbols).

NMF\_CREDIT—Formats the bignum with the locale-specific credit symbols whenever the bignum is negative.

NMF\_CURRENCY—Formats the bignum string with the locale-specific currency symbol.

**NMF\_DECIMAL(*decimal*)**—Formats the bignum string with *decimal* number of decimal places to the right of the decimal point. Decimal places from 0 to 8 are supported by default. If more decimal places are desired, modify the value of the *NUMBER\_DECIMAL* macro as described above.

**NMF\_DIGITS(*digits*)**—Formats the bignum string with *digits* number of digits to the left of the decimal point. Digits from 0 to 30 are supported. If more digits are desired, modify the value of the *NUMBER\_WHOLE* macro as described above.

**NMF\_NO\_FLAGS**—Does not associate any special flags with the *ZIL\_BIGNUM* class object. This flag should not be used in conjunction with any other NMF flags.

**NMF\_PERCENT**—Formats the bignum with the percent symbol.

- *decimalString<sub>in</sub>* is a pointer to the decimal character to be used in formatting the decimal number.
- *signStr<sub>in</sub>* is a pointer to the sign character to be used in formatting the bignum.

## Example

```
#include <ui_gen.hpp>

ExampleFunction( )

{

    char string2[40];
    number.Export(string2, NMF_NO_FLAGS);
}

```

## ZIL\_BIGNUM::floor

### Syntax

```
friend ZIL_BIGNUM &floor(const ZIL_BIGNUM &number);
```



## 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 a ZIL\_BIGNUM object containing the floor value of the ZIL\_BIGNUM value passed in. The floor of a bignum is considered to be the largest integer value that is not greater than *number*.

- *returnValue<sub>out</sub>* is a ZIL\_BIGNUM object containing the floor value of *number*.
- *number<sub>in</sub>* is a ZIL\_BIGNUM object for which the floor value is desired.

## Example

```
#include <ui_gen.hpp>

ExampleFunction( )
{
    ZIL_BIGNUM *firstValue;
    ZIL_BIGNUM secondValue("100.6");

    firstValue=floor(secondValue);
}
```

## ZIL\_BIGNUM::GetLocale

### Syntax

```
const ZIL_LOCALE *GetLocale(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 *myLocale*, the ZIL\_LOCALE that provides formatting information.

- *returnValue<sub>out</sub>* is a pointer to *myLocale*.

## ZIL\_BIGNUM::Import

### Syntax

```
#include <ui_gen.hpp>

NMI_RESULT Import(ZIL_IBIGNUM value);
    or
NMI_RESULT Import(ZIL_RBIGNUM value);
    or
NMI_RESULT Import(const ZIL_BIGNUM &number);
    or
NMI_RESULT Import(const ZIL_ICHAR *string,
    const ZIL_ICHAR *decimalString = ZIL_NULLP(ZIL_ICHAR),
    const ZIL_ICHAR *signString = 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

These overloaded functions set the numerical information with a function-specific numeric value.

The first two overloaded functions copy the bignum information from the *value* argument.

- *returnValue<sub>out</sub>* is NMI\_OK if the conversion was successful. Otherwise, *returnValue* is NMI\_OUT\_OF\_RANGE and the bignum object will not be modified.

- *value<sub>in</sub>* is a numeric value. The following values are supported:

ZIL\_IBIGNUM—A type which will be a signed int of at least 32 bits, depending on the environment being compiled for. For an environment that uses 32-bit words or smaller, the value can be between -2,147,483,648 and 2,147,483,647 (32 bits, signed), inclusive. For environments that use 64-bit words, the value can be between -9,223,372,036,854,775,809 and 9,223,372,036,854,775,808 (64 bits, signed), inclusive.

ZIL\_RBIGNUM—A double precision floating point bignum.

The third overloaded function copies the bignum information from the *number* argument.

- *returnValue<sub>out</sub>* is NMI\_OK if the conversion was successful. Otherwise, *returnValue* is NMI\_OUT\_OF\_RANGE and the bignum object will not be modified.
- *number<sub>in</sub>* is a ZIL\_BIGNUM reference variable. The value in *number* is copied into the bignum object.

The last overloaded function sets the ZIL\_BIGNUM information according to the string argument.

- *returnValue<sub>out</sub>* is NMI\_OK if the conversion was successful. Otherwise, *returnValue* is NMI\_OUT\_OF\_RANGE and the bignum object will not be modified.
- *string<sub>in</sub>* is a pointer to a string that represents a bignum in character form.
- *decimalString<sub>in</sub>* is a pointer to the decimal character to be used in formatting the decimal number.
- *signString<sub>in</sub>* is a pointer to the sign character to be used in formatting the bignum.

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_ICHAR *string = "100";
    ZIL_BIGNUM number;
    number.Import(string);
}
```

## ZIL\_BIGNUM::round

### Syntax

```
friend ZIL_BIGNUM &round(const ZIL_BIGNUM &number, int places = 0);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function returns a pointer to a ZIL\_BIGNUM object containing the rounded value of the ZIL\_BIGNUM value passed in.

- *returnValue<sub>out</sub>* is a ZIL\_BIGNUM object containing the value of *number* rounded to the *places* decimal place.
- *number<sub>in</sub>* is the ZIL\_BIGNUM value to be rounded.
- *places<sub>in</sub>* determines how many decimal places to round *number*. For example, if *places* were 1, the value 100.163 would be rounded to 100.2. If *places* were -1, the value 123.789 would be rounded to 120. The default value, 0, causes *number* to be rounded to a whole number.

### Example

```
void UIW_INTL_CURRENCY::SetCountryCode(int _countryTableEntry)
{
    //Do Currency translation.
    ZIL_BIGNUM *amount = DataGet();
    ZIL_RBIGNUM value;
    amount->Export(&value);

    value *= _currency[_countryTableEntry][_countryTableEntry];
    amount->Import(value);
    #amount = round(*amount, 2);

    countryTableEntry = _countryTableEntry;
    DataSet(amount);
}
```

## ZIL **BIGNUM::SetLocale**

### Syntax

```
void SetLocale(const ZIL_ICHAR *localeName);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function sets the locale to be used by the object. The locale information for the object will be loaded and the object's *myLocale* member will be updated to point to the new ZIL\_LOCALE object. By default, the object uses the locale identified in the **LOC\_DEF.CPP** file, which compiles into the library. (If a different default locale is desired, simply copy a **LOC\_<ISO>.CPP** file from the ZINC\SOURCE\INTL directory to the \ZINC\SOURCE directory, and rename it to **LOC\_DEF.CPP** before compiling the library.) The locale information is loaded from the **I18N.DAT** file, so it must be shipped with your application.

- *localeName<sub>m</sub>* is the two-letter ISO country code identifying which locale information the object should use.

## ZIL **BIGNUM::truncate**

### Syntax

```
friend ZIL_BIGNUM &truncate(const ZIL_BIGNUM &number, int places = 0);
```

### Portability

This function is available on the following environments:

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

## Remarks

This function returns a pointer to a ZIL\_BIGNUM object containing the truncated value of the ZIL\_BIGNUM value passed in.

- *returnValue<sub>out</sub>* is a ZIL\_IBIGNUM object containing the value of *number* after being truncated to *places* decimal places.
- *number<sub>in</sub>* is the ZIL\_BIGNUM value to be truncated.
- *places<sub>in</sub>* specifies to which digit to truncate *number*. For example, if *places* were 1, the value 100.163 would be truncated to 100.1. If *places* were -1, the value 123.789 would be truncated to 120. The default value, 0, causes *number* to be truncated to the decimal point.

## Example

```
#include <ui_gen.hpp>

ExampleFunction( )
{
    ZIL_BIGNUM *firstValue;
    ZIL_BIGNUM secondValue("100.6");

    firstValue = truncate(secondValue);
}
```

## ZIL\_BIGNUM-operator =

### Syntax

```
#include <ui_gen.hpp>

ZIL_BIGNUM &operator = (const ZIL_BIGNUM &number);
```

### Portability

This function is available on the following environments:

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

## Remarks

This operator overload assigns the value of another ZIL\_BIGNUM object specified by *number* to the ZIL\_BIGNUM object.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_BIGNUM object after its value has been modified. This pointer is returned so that the operator may be used in a statement containing other operations.
- *number<sub>in</sub>* is a ZIL\_BIGNUM object containing the source value to be assigned.

## Example

```
#include <ui_gen.hpp>

ExampleFunction( )
{
    ZIL_BIGNUM sourceValue("100");
    ZIL_BIGNUM targetValue;

    targetValue = sourceValue;
}
```

## ZIL\_BIGNUM-operator +

### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_BIGNUM &operator + (const ZIL_BIGNUM &number);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload adds the value of another ZIL\_BIGNUM object specified by *number* to the ZIL\_BIGNUM object.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_BIGNUM object after its value has been modified. This pointer is returned so that the operator may be used in a statement containing other operations.
- *number<sub>in</sub>* is a bignum object containing the value to be added to the ZIL\_BIGNUM object.

## Example

```
#include <ui_gen.hpp>
ExampleFunction( )
{
    ZIL_BIGNUM firstValue;
    ZIL_BIGNUM secondValue("200");

    firstValue.Import("100");
    secondValue = secondValue + firstValue;
}

```

## ZIL\_BIGNUM::operator -

### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_BIGNUM &operator - (const ZIL_BIGNUM &number);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload subtracts the value of another ZIL\_BIGNUM object specified by *number* from the ZIL\_BIGNUM object.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_BIGNUM object after its value has been modified. This pointer is returned so that the operator may be used in a statement containing other operations.



- *number<sub>in</sub>* is another ZIL\_BIGNUM object containing the value to be subtracted from the ZIL\_BIGNUM object.

## Example

```
#include <ui_gen.hpp>

ExampleFunction( )
{
    ZIL_BIGNUM firstValue;
    ZIL_BIGNUM secondValue("200");

    firstValue.Import("100");
    secondValue = secondValue - firstValue;
}

```

## ZIL\_BIGNUM-operator \*

### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_BIGNUM &operator * (const ZIL_BIGNUM &number)\
```

### Portability

This function is available on the following environments:

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

### Remarks

The operator overload multiplies the value of another ZIL\_BIGNUM object specified by *number* by the ZIL\_BIGNUM object.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_BIGNUM object after its value has been modified. This pointer is returned so that the operator may be used in a statement containing other operations.
- *number<sub>in</sub>* is another ZIL\_BIGNUM object containing the value to be multiplied by the ZIL\_BIGNUM object.

## Example

```
#include <ui_gen.hpp>

ExampleFunction( )
{
    ZIL_BIGNUM firstValue;
    ZIL_BIGNUM secondValue("200");

    firstValue.Import("100");
    secondValue = secondValue *firstValue;
}

```

## ZIL\_BIGNUM::operator ++

### Syntax

```
#include <ui_gen.hpp>

ZIL_BIGNUM &operator ++ (void);

```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload increments the ZIL\_BIGNUM object's value by one.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_BIGNUM object after its value has been modified. This pointer is returned so that the operator may be used in a statement containing other operations.

## Example

```
#include <ui_gen.hpp>

ExampleFunction(ZIL_BIGNUM &number)

```

```
    number++;  
}
```

## ZIL\_BIGNUM::operator --

### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_BIGNUM &operator -- (void);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload decrements the ZIL\_BIGNUM object's value by one.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_BIGNUM object after its value has been modified. This pointer is returned so that the operator may be used in a statement containing other operations.

### Example

```
#include <ui_gen.hpp>  
ExampleFunction(ZIL_BIGNUM &number)  
{  
  
    number--;  
}
```

## ZIL\_BIGNUM::operator +=

### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_BIGNUM &operator += (const ZIL_BIGNUM &number);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload adds the value of another ZIL\_BIGNUM object, specified by *number*, to the ZIL\_BIGNUM object and copies the result back into the ZIL\_BIGNUM object.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_BIGNUM object after its value has been modified. This pointer is returned so that the operator may be used in a statement containing other operations.
- *number<sub>in</sub>* is another ZIL\_BIGNUM object containing the value to be added to the ZIL\_BIGNUM object.

### Example

```
#include <ui_gen.hpp>

ExampleFunction( )
{
    ZIL_BIGNUM firstValue;
    ZIL_BIGNUM secondValue;

    firstValue.Import ("100");
    secondValue.Import ("200");
    secondValue += firstValue;
}
```

## **ZIL\_BIGNUM::operator !=**

### **Syntax**

```
#include <ui_gen.hpp>
```

```
ZIL_BIGNUM &operator -= (const ZIL_BIGNUM &number);
```

### **Portability**

This function is available on the following environments:

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

### **Remarks**

This operator overload subtracts the value of another ZIL\_BIGNUM object, specified by *number*, from the ZIL\_BIGNUM object and copies the result back into the ZIL\_BIGNUM object.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_BIGNUM object after its value has been modified. This pointer is returned so that the operator may be used in a statement containing other operations.
- *number<sub>in</sub>* is another ZIL\_BIGNUM object containing the value to be subtracted from the ZIL\_BIGNUM object.

### **Example**

```
#include <ui_gen.hpp>

ExampleFunction( )
{
    ZIL_BIGNUM firstValue;
    ZIL_BIGNUM secondValue;

    firstValue.Import ("100");
    secondValue.Import ("200");
    secondValue -= firstValue;
}

```

## ZIL\_BIGNUM::operator ==

### Syntax

```
#include <ui_gen.hpp>

int operator == (const ZIL_BIGNUM &number)\
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload determines if the value of the ZIL\_BIGNUM object is equal to the value of the ZIL\_BIGNUM object specified by *number*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_BIGNUM object is equal to *number*. Otherwise, *returnValue* is FALSE.
- *number<sub>in</sub>* is the other ZIL\_BIGNUM object to be compared.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_BIGNUM governmentRevenue("10389230299.49");
    ZIL_BIGNUM governmentSpending("378321783443199.81");

    if (governmentRevenue == governmentSpending)
        printf("Budget is balanced?\n");
    else if (governmentRevenue < governmentSpending)
        printf("Big deal, this is normal.\n");
    else
        printf("Must be a computer error!\n");
}
```

## **ZIL\_BIGNUM::operator !=**

### **Syntax**

```
#include <ui_gen.hpp>

int operator != (const ZIL_BIGNUM &number);
```

### **Portability**

This function is available on the following environments:

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

### **Remarks**

This operator overload determines if the value of the ZIL\_BIGNUM object is not equal to the value of the ZIL\_BIGNUM object specified by *number*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_BIGNUM object is not equal to *number*. Otherwise, *returnValue* is FALSE.
- *number<sub>in</sub>* is the other ZIL\_BIGNUM object to be compared.

### **Example**

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_BIGNUM totalDays("400");
    ZIL_BIGNUM daysPerYear("365");

    if (totalDays != daysPerYear)
    {
        if (totalDays < daysPerYear)
            printf("Less than one year has passed.\n");
        else
            printf("More than one year has passed.\n");
    }
}
```

## ZIL\_BIGNUM::operator >

### Syntax

```
#include <ui_gen.hpp>

int operator > (const ZIL_BIGNUM &number)
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload determines whether the value of the ZIL\_BIGNUM object is greater than the value of the ZIL\_BIGNUM object specified by *number*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_BIGNUM object is greater than *number*. Otherwise, *returnValue* is FALSE.
- *number<sub>in</sub>* is the other ZIL\_BIGNUM object to be compared.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_BIGNUM governmentRevenue("10389230299.49");
    ZIL_BIGNUM governmentSpending("378321783443199.81");

    if (governmentRevenue == governmentSpending)
        printf("Budget is balanced?\n");
    else if (governmentRevenue > governmentSpending)
        printf("Must be a computer error!\n");
    else
        printf("Big deal, this is normal.\n");
}
```



## ZIL\_BIGNUM::operator >=

### Syntax

```
#include <ui_gen.hpp>
```

```
int operator >= (const ZIL_BIGNUM &number);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload determines whether the value of the ZIL\_BIGNUM object is greater than or equal to the value of the ZIL\_BIGNUM object specified by *number*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_BIGNUM object is greater than or equal to *number*. Otherwise, *returnValue* is FALSE.
- *number<sub>in</sub>* is the other ZIL\_BIGNUM object to be compared.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_BIGNUM totalDays("400");
    ZIL_BIGNUM daysPerYear("365");

    if (totalDays >= daysPerYear)
        printf("One year has passed.\n");
    else
        printf("Less than one year has passed.\n");
}
```

## ZIL\_BIGNUM::operator <

### Syntax

```
#include <ui_gen.hpp>

int operator < (const ZIL_BIGNUM &number)\
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload determines whether the value of the ZIL\_BIGNUM object is less than the value of the ZIL\_BIGNUM object specified by *number*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_BIGNUM object is less than *number*. Otherwise, *returnValue* is FALSE.
- *number<sub>in</sub>* is the other ZIL\_BIGNUM object to be compared.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_BIGNUM governmentRevenue("10389230299.49");
    ZIL_BIGNUM governmentSpending("378321783443199.81");

    if (governmentRevenue == governmentSpending)
        printf("Budget is balanced?\n");
    else if (governmentRevenue < governmentSpending)
        printf("What's new?!\n");
    else
        printf("Must be a computer error!\n");
}
```

## ZIL\_BIGNUM::operator <=

### Syntax

```
#include <ui_gen.hpp>
```

```
int operator <= (const ZIL_BIGNUM &number);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload determines whether the value of the ZIL\_BIGNUM object is less than or equal to the value of the ZIL\_BIGNUM object specified by *number*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_BIGNUM object is less than or equal to *number*. Otherwise, *returnValue* is FALSE.
- *number<sub>in</sub>* is the other ZIL\_BIGNUM object to be compared.

### Example

```
#include <ui_gen.hpp>
ExampleFunction()
{
    ZIL_BIGNUM totalDays("400");
    ZIL_BIGNUM daysPerYear("365");

    if (totalDays <= daysPerYear)
        printf("Less than one year has passed.\n");
    else
        printf("One year has passed.\n");
}
```

## CHAPTER 50 - ZIL\_BITMAP\_ELEMENT

The `ZIL_BITMAP_ELEMENT` structure is used by the `ZIL_DECORATION` class to provide the bitmap decorations for objects. An object's decorations are those bitmaps or characters that are used to draw an image on the object. The decorations typically include a graphical image, or bitmap, for use in graphics mode and a textual image, or character string, for use in text mode. Most environments don't require these decorations since the operating system typically provides them. OpenZinc does all the drawing in DOS and Curses, however, so these environments use decorations extensively. An example of where a decoration would be used is the maximize button. In graphics mode, it typically has a small up-arrow bitmap. In text mode, though, it usually displays a left bracket, an up-arrow character, and a right-bracket; all text characters, of course. This class maintains the bitmap images. See "Chapter 71—`ZIL_TEXT_ELEMENT`" for information on the text strings used for decorations.

The `ZIL_BITMAP_ELEMENT` class is declared in `UI_GEN.HPP`. Its public and protected members are:

```
Struct ZIL_EXPORT_CLASS ZIL_BITMAP_ELEMENT
{
    ZIL_UINT8 *bitmap;
    ZIL_NUMBERID numberID;
    ZIL_ICHAR stringID[ZIL_STRINGID_LEN];
};
```

### General Members

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

- *bitmap* is the bitmap array maintained by the `ZIL_BITMAP_ELEMENT`.
- *numberID* is a numeric value used to identify the `ZIL_BITMAP_ELEMENT`.
- *stringID* is a string value used to identify the `ZIL_BITMAP_ELEMENT`.



## CHAPTER 51 - ZIL\_DATE

The ZIL\_DATE class is a lower-level class used to store and manipulate date values. It is not a window object. See "Chapter 5—UIW\_DATE" of *Programmer's Reference Volume 2* for information about the date window object.

**NOTE:** The **DayOfWeek**, **DaysInMonth** and **DaysInYear** functions may return questionable values for dates before 1753 due to the switch from the Julian calendar to the Gregorian calendar.

The ZIL\_DATE class is declared in **UI\_GEN.HPP**. Its public and protected members are:

```
class ZIL_EXPORT_CLASS ZIL_DATE : public ZIL_UTIME
public:
    ZIL_DATE(void);
    ZIL_DATE(const ZIL_DATE &date);
    ZIL_DATE(int year, int month, int day);
    ZIL_DATE(const ZIL_ICHAR *string, DTF_FLAGS dtFlags = DTF_NO_FLAGS);
    ZIL_DATE(int packedDate);
    int DayOfWeek(void);
    int DaysInMonth(void);
    int DaysInYear(void);
    void Export(int *year, int *month, int *day,
               int *dayOfWeek = ZIL_NULLP(int));
    void Export(ZIL_ICHAR *string, DTF_FLAGS dtFlags);
    void Export(int *packedDate);
    DTI_RESULT Import(void);
    DTI_RESULT Import(const ZIL_DATE &date);
    DTI_RESULT Import(int year, int month, int day);
    DTI_RESULT Import(const ZIL_ICHAR *string,
                     DTF_FLAGS dtFlags = DTF_NO_FLAGS);
    DTI_RESULT Import(int packedDate);

    ZIL_INT32 operator=(ZIL_INT32 days);
    ZIL_INT32 operator=(const ZIL_DATE &date);
    ZIL_INT32 operator+(ZIL_INT32 days);
    ZIL_INT32 operator+(const ZIL_DATE &date);
    ZIL_INT32 operator-(ZIL_INT32 days);
    ZIL_INT32 operator-(const ZIL_DATE &date);
    ZIL_INT32 operator++(void);
    ZIL_INT32 operator--(void);
    void operator+=(ZIL_INT32 days);
    void operator-=(ZIL_INT32 days);
    int operator==(const ZIL_DATE &date);
    int operator!=(const ZIL_DATE &date);
    int operator>(const ZIL_DATE &date);
    int operator>=(const ZIL_DATE &date);
    int operator<(const ZIL_DATE &date);
    int operator<=(const ZIL_DATE &date);

    void SetBasis(int _basisYear);
    int GetBasis();
};
```

## General Members

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

### **ZIL\_DATE::ZIL\_DATE**

#### **Syntax**

```
#include <ui_gen.hpp>

ZIL_DATE(void);
    or
ZIL_DATE(const ZIL_DATE &date);
    or
ZIL_DATE(int year, int month, int day);
    or
ZIL_DATE(const ZIL_ICHAR *string, DTF_FLAGS dtFlags = DTF_NO_FLAGS);
    or

ZIL_DATE(int packedDate);
```

#### **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 ZIL\_IDATE object.

The first overloaded constructor takes no arguments. It sets the date information according to the system's date.

The second overloaded constructor is a copy constructor that takes a previously constructed ZIL\_IDATE object to specify the default date.

- *date<sub>in</sub>* is a reference pointer to a previously constructed ZIL\_DATE object.

The third overloaded constructor uses integer arguments to specify the default date.

- *year<sub>in</sub>* is the year. This argument must be either 0, if no year value is to be used with the date, or a value in a range from 100 to 32,767.
- *month<sub>in</sub>* is the month. This argument must be either 0, if no month value is to be used with the date, or a value in a range from 1 (January) to 12 (December).
- *day<sub>in</sub>* is the day. This argument must be either 0, if no day value is to be used with the date, or a value in a range from 1 to 31 that should be valid for the specified month and year.

The fourth overloaded constructor uses a string argument to specify the default date. The following algorithm is used to determine the proper order and meaning of date values:

1—Any number greater than 31 is assumed to be the year.

2—If the number is less than 100, the basis year is added to the value. See `ZIL_DATE::SetBasis()` below for information about the basis year. Year values below 100 are not allowed in the `ZIL_DATE` class.

3—Any number between 13 and 31 is assumed to be the day. In ambiguous situations where both the day and month values are less than 13, the country code date format (e.g., `DTF_US_FORMAT`, `DTF_ASIAN_FORMAT`) is used to decide the order of date values.

- *string<sub>in</sub>* is a string that contains the date information.
- *dtFlags<sub>in</sub>* specifies how to interpret the date string. The following flags (declared in `UI_GEN.HPP`) override the country dependent information (supplied by the operating system):

**DTF\_EUROPEAN\_FORMAT**—Forces the date to be interpreted in the European format (i.e., *day/month/year*), regardless of the default country information.

**DTF\_ASIAN\_FORMAT**—Forces the date to be interpreted in the Asian format (i.e., *year/month/day*), regardless of the default country information.

**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.

**DTF\_NO\_FLAGS**—Does not associate any special flags with the ZIL\_DATE object. In this case, the string will be interpreted using the default country information. This is the default argument if no other argument is provided. This flag should not be used in conjunction with any other DTF flags.

**DTF\_SYSTEM**—Sets the date value according to the system date if the string is blank or NULL. For example, if the DTF\_SYSTEM flag were set and a NULL string value was specified, the date would be set to the system date.

**DTF\_US\_FORMAT**—Forces the date to be interpreted in the U.S. format (i.e., *month/day/year*), regardless of the default country information.

The fifth overloaded constructor uses a packed integer argument to specify the default date.

- *packedDate<sub>in</sub>* is a packed representation of the date (whose format is the same as the MS-DOS file dates). This argument is packed according to the following bit pattern:

bits 0-4 specify the day,  
bits 5-8 specify the month, and  
bits 9-15 specify the year minus 1980 (e.g., a value of 5 means 1985).

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_DATE date1;
    ZIL_DATE date2(1990, 1, 1);
    ZIL_DATE *date3 = new ZIL_DATE("Jan. 1, 1990");
    ZIL_DATE *date4 = new ZIL_DATE(date1);

    delete date4;
    delete date3;
    // The destructors for date1 and date2 are automatically called
    // when the scope of this function ends.
}
```

## ZIL\_DATE::DayOfWeek

### Syntax

```
#include <ui_gen.hpp>

int DayOfWeek(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 numerical value of the day of the week (Sunday = 1, Monday = 2, . . . Saturday = 7) for the ZIL\_DATE object.

**NOTE:** `DayOfWeek()` may return questionable values for dates before 1753 due to the switch from the Julian calendar to the Gregorian calendar.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_DATE date;

    date.dayTable =
    {
        "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
        "Friday", "Saturday"
    };

    // Print the current day of week.
    printf("Today is %s.\n", date.dayTable[date.DayOfWeek() - 1]);
}
```

## ZIL\_DATE::DaysInMonth

### Syntax

```
#include <ui_gen.hpp>

int DaysInMonth(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 number of days in the month specified by the ZIL\_DATE object. For example, if the date were December 15, 1993, **DaysInMonth** would return 31.

**NOTE:** **DaysInMonth()** may return questionable values for dates before 1753 due to the switch from the Julian calendar to the Gregorian calendar.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    // Print the number of days in the current month.
    ZIL_DATE date;
    printf("This month has %d days.\n", date.DaysInMonth());
}
```

## ZIL\_DATE::DaysInYear

### Syntax

```
#include <ui_gen.hpp>

int DaysInYear( 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 number of days in the year specified by the `ZIL_DATE` object. For example, if the date were January 15, 1992, `DaysInYear()` would return 366 (i.e., 1 extra day for leap year).

**NOTE:** `DaysInYear()` may return questionable values for dates before 1753 due to the switch from the Julian calendar to the Gregorian calendar.

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    // Print the number of days in the year.
    ZIL_DATE date;
    printf("This year has %d days.\n", date.DaysInYear());
}
```

## ZIL\_DATE::Export

### Syntax

```
#include <ui_gen.hpp>

void Export(int *year, int *month, int *day, int *dayOfWeek = ZIL_NULLP(int));
    or
void Export(ZIL_ICHAR *string, DTF_FLAGS dtFlags);
    or
void Export(int *packedDate);
```

## 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 obtain the value of the ZIL\_DATE object.

The first overloaded function returns date information through four integer arguments.

- *year<sub>out</sub>* is a pointer to the variable that is to contain the year. If this argument is NULL, no year information is returned. If there is no year associated with the date, this argument will be 0. Otherwise, this argument will be a value within the range 100 to 32,767.
- *month<sub>out</sub>* is a pointer to the variable that is to contain the month. If this argument is NULL, no month information is returned. If there is no month associated with the date, this argument will be 0. Otherwise, this argument will be a value within the range 1 (January) to 12 (December).
- *day<sub>out</sub>* is a pointer to the variable that is to contain the day. If this argument is NULL, no day information is returned. If there is no day associated with the date, this argument will be 0. Otherwise, this argument will be a value within the range 1 to 31.
- *dayOfWeek<sub>out</sub>* is a pointer to the variable that is to contain the day-of-week. If this argument is NULL, no day-of-week information is returned. If the year, month and day values are all present in the date, this argument will be a value within the range 1 (Sunday) to 7 (Saturday). Otherwise, this argument will be 0.

The second overloaded function returns the date information through the *string* argument.

- *string<sub>in</sub>* is a pointer to a string that gets the formatted date. This string must be long enough to contain the date.
- *dtFlags<sub>in</sub>* specifies how the return date should be formatted. The following flags (declared in UI\_GEN.HPP) override the country dependent information (supplied by the operating system):

<p><b>DTF_ALPHA_MONTH</b>—Causes the month name to be spelled-out, as opposed to being represented numerically.</p>	<p>March 28, 1990 December 4, 1980 January 3, 2003</p>
<p><b>DTF_DASH</b>—Separates the date fields with a dash, regardless of the default country date separator.</p>	<p>3-28-1990 12-04-1980 1-3-2003</p>
<p><b>DTF_DAY_OF_WEEK</b>—Causes a spelled-out day-of-week to be shown in the date.</p>	<p>Monday May 4, 1992 Friday Dec. 5, 1980 Sunday Jan. 4, 2003</p>
<p><b>DTF_EUROPEAN_FORMAT</b>—Forces the date to be formatted in the European format (i.e., <i>day/month/year</i>), regardless of the default country information.</p>	<p>28/3/1990 4 December, 1980 3 Jan., 2003</p>
<p><b>DTF_ASIAN_FORMAT</b>—Forces the date to be formatted in the Asian format (i.e., <i>year/month/day</i>), regardless of the default country information.</p>	<p>1990/3/28 1980 December 4 2003 Jan. 3</p>
<p><b>DTF_MILITARY_FORMAT</b>—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 <i>day month year</i> where <i>month</i> is either a 3-letter abbreviated word and <i>year</i> is a two-digit year value (if the <b>DTF_SHORT_YEAR</b> or <b>DTF_SHORT_MONTH</b> flags are set) or <i>month</i> is spelled-out and <i>year</i> 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.</p>	<p>(air force style- default) 4 Jul 91 4 July 1991</p>
<p><b>DTF_NO_FLAGS</b>—Does not associate any special flags with the <b>Export( )</b> function. 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.</p>	<p>(European format) 4 December 1989 23 June 2000  (Asian format) 1989 December 4 2000 June 23</p>

<b>DTF_SHORT_DAY</b> —Adds an abbreviated day-of-week to the date.	Wed. March 28, 1990 Thurs. Dec. 4, 1980 Sat. January 3, 2003
<b>DTF_SHORT_MONTH</b> —Adds an abbreviated month name to the date.	Mar. 28, 1990 Dec. 4, 1980 Jan. 3, 2003
<b>DTF_SHORT_YEAR</b> —Forces the year to be formatted as a two-digit value.	3/28/90 December 4, 80 Jan. 3, 89
<b>DTF_SLASH</b> —Separates the date fields with a slash, regardless of the default country date separator.	3/28/90 12/04/1900 1/3/2003
<b>DTF_SYSTEM</b> —Uses the system date.	3/28/90 12/04/1980 1/3/2003
<b>DTF_UPPER_CASE</b> —Converts the alphabetic date characters to upper-case.	MARCH 28, 1990 DEC. 4, 1980 SATURDAY JAN 3, 2003
<b>DTF_US_FORMAT</b> —Forces the date to be formatted in the U.S. format (i.e., <i>month/day/year</i> ), regardless of the default country information.	March 28, 1990 12/4/1980 Jan 3, 2003
<b>DTF_ZERO_FILL</b> —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

The third overloaded function returns date information through a packed integer.

- *packedDate<sub>out</sub>* is a packed representation of the date (whose format is the same as the MS-DOS file dates). This argument is packed according to the following bit pattern:

bits 0-4 specify the day,  
bits 5-8 specify the month, and  
bits 9-15 specify the year minus 1980 (e.g., a value of 5 means 1985).

## Example

```
#include <ui_gen.hpp>
ExampleFunction()
{
    ZIL_DATE date; // Initialize a system date.
}
```

```

// Print out the date in various forms.
int year, month, day;
date.Export(&year, &month, &day);
printf("Integer date value: year-%d, month-%d, day-%d\n",
       year, month, day);
char stringDate[128];
date.Export(stringDate, DTF_NO_FLAGS);
printf("String date value: %s", stringDate);

// The destructor for date is automatically called when the
// scope of this function ends.
}

```

## ZIL\_DATE::GetBasis

### Syntax

```

#include <ui_gen.hpp>

int GetBasis( );

```

### 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 year being used as a basis for date manipulation by the ZIL\_DATE object.

- *returnValue<sub>out</sub>* is the year used as the basis. The basis year identifies the century to be used to resolve two-digit year abbreviations. For example, if the basis year is 1800, then a year of 63 is resolved to be 1863. The basis year by default is obtained from the operating system.



## ZIL\_DATE::Import

### Syntax

```
#include <ui_gen.hpp>

DTI_RESULT Import(void);
    or
DTI_RESULT Import(const ZIL_DATE &date);
    or
DTI_RESULT Import(int year, int month, int day);
    or
DTI_RESULT Import(const ZIL_ICHAR *string,
    DTF_FLAGS dtFlags = DTF_NO_FLAGS);
    or
DTI_RESULT Import(int packedDate);
```

### 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 set the value of the ZIL\_IDATE object.

The first overloaded function sets the date information according to the system date.

- *returnValue<sub>out</sub>* is the result of the import operation. *returnValue* can have one of the following values:

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 **Ja**an 28, 1992" or "**Tuesday** 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").

The second overloaded function copies the date information from the *date* reference argument.

- *returnValue<sub>out</sub>* is the result of the import operation. See the first function for possible values.
- *date<sub>in</sub>* is a reference pointer to a previously constructed date.

The third overloaded function sets the date information according to specified integer arguments.

- *returnValue<sub>out</sub>* is the result of the import operation. See the first function for possible values.
- *year<sub>in</sub>* is the year. This argument must be 0 if no year value is to be used with the date, or a value in the range 100 to 32,767.
- *month<sub>in</sub>* is the month. This argument must be 0 if no month value is to be used with the date, or a value in the range 1 (January) to 12 (December).
- *day<sub>in</sub>* is the day. This argument must be 0 if no day value is to be used with the date, or a value in the range 1 to 31 that should be valid for the specified month and year.

The fourth overloaded function sets the date using information passed in a string. The following algorithm is used to determine the proper order and meaning of date values:

1—Any number greater than 31 is assumed to be the year.

**2**—If the number is less than **100**, the basis year is added to the value. See **ZIL\_DATE::SetBasis()** below for information about the basis year. Year values below **100** are not allowed in the **ZIL\_DATE** class.

**3**—Any number between 13 and 31 is assumed to be the day. In ambiguous situations where both the day and month values are less than 13, the country code date format (e.g., **DTF\_US\_FORMAT**, **DTF\_ASIAN\_FORMAT**) is used to determine the order of date values.

- *returnValue<sub>out</sub>* is the result of the import operation. See the first function for possible values.
- *string<sub>in</sub>* is a pointer to the date string. If this is an empty string (i.e., ""), the **ZIL\_DATE** will be set to "blank." Passing a blank **ZIL\_IDATE** to the **UIW\_DATE::DataSet()** function will cause the date field to be displayed as blank space. See the **DataSet** section of "Chapter 5—**UIW\_DATE**" in *Programmer's Reference Volume 2* for more information.
- *dtFlags<sub>in</sub>* specifies how the date string should be interpreted. The following flags (declared in **UI\_GEN.HPP**) override the country dependent information (supplied by the operating system):

**DTF\_EUROPEAN\_FORMAT**—Forces the date to be interpreted in the European format (i.e., *day/month/year*), regardless of the default country information.

**DTF\_ASIAN\_FORMAT**—Forces the date to be interpreted in the Asian format (i.e., *year/month/day*), regardless of the default country information.

**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.

**DTF\_NO\_FLAGS**—Does not associate any special flags with the **ZIL\_DATE** object. In this case, the string will be interpreted using the default country information. This flag should not be used in conjunction with any other **DTF** flags.

**DTF\_SYSTEM**—Sets the date value according to the system date if the string is blank or NULL. For example, if the DTF\_SYSTEM flag were set and a NULL string value was specified, the date would be set to the system date.

**DTF\_US\_FORMAT**—Forces the date to be interpreted in the U.S. format (i.e., *month/day/year*), regardless of the default country information.

The  `fifth`  overloaded function sets the date information through a packed integer argument.

- `returnValueout`  is the result of the import operation. See the first function for possible values.
- `packedDatein`  is a packed representation of the date (whose format is the same as the MS-DOS file dates). This argument is packed according to the following bit pattern:

bits 0-4 specify the day,  
bits 5-8 specify the month, and  
bits 9-15 specify the year minus 1980 (e.g., a value of 5 means 1985).

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_DATE date; // Initialize a system date.

    // Import the date in various forms and print out the results.
    char stringDate[128];
    date.Import(1990, 1, 1);
    date.Export(stringDate, DTF_NO_FLAGS);
    printf("String date value: %s\n", stringDate);
    date.Import("1-1-1990", DTF_NO_FLAGS);
    date.Export(stringDate, DTF_MILITARY_FORMAT);
    printf("String date value: %s\n", stringDate);

    // The destructor for date is automatically called when the
    // scope of this function ends.
}
```

## ZIL\_DATE::SetBasis

### Syntax

```
#include <ui_gen.hpp>

void SetBasis(int _basisYear);
```

## 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 year being used as a basis for date manipulation by the `ZIL_DATE` object.

- `_basisYearin` is the year to be used as the basis. The basis year identifies the century to be used to resolve two-digit year abbreviations. For example, if the basis year is 1800, a year of 63 is resolved to be 1863. The basis year by default is obtained from the operating system.

## ZIL\_DATE::operator =

### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_INT32 operator=(ZIL_INT32 days);
```

*or*

```
ZIL_INT32 operator=(const ZIL_DATE &Date);
```

## Portability

This function is available on the following environments:

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

## Remarks

The first operator overload assigns the value specified by `days` to the `ZIL_DATE` object.

- `returnValueout` is the number of days in the resulting date. This raw value is returned so that the operator may be used in a statement containing other operations.

- *days<sub>in</sub>* is the date, given in the number of days, to be assigned to the ZIL\_DATE object.

The second operator overload assigns the value specified by *date* to the ZIL\_DATE object.

- *returnValue<sub>out</sub>* is the number of days in the resulting date. This raw value is returned so that the operator may be used in a statement containing other operations.
- *date<sub>in</sub>* is the date to be assigned to the ZIL\_DATE object.

### Example

```
#include <ui_gen.hpp>

AddOneWeek(ZIL_DATE currentDate, ZIL_DATE &nextWeek)
{
    ZIL_INT32 oneWeek = 7;
    // Adding 1 week to the current date gives the next week,
    nextWeek = currentDate + oneWeek;
}
```

## ZIL\_DATE::operator +

### Syntax

```
#include <ui_gen.hpp>

ZIL_INT32 operator+(ZIL_INT32 days);
or
ZIL_INT32 operator+(const ZIL_DATE &date);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload adds the value *days* to the ZIL\_DATE object.

- *returnValue<sub>out</sub>* is the number of days resulting from the addition operation. This raw value is returned so that the operator may be used in a statement containing other operations.
- *days<sub>in</sub>* is the number of days to be added to the ZIL\_DATE object.

The second operator overload adds the date contained in *date* to the ZIL\_DATE object.

- *returnValue<sub>out</sub>* is the number of days resulting from the addition operation. This raw value is returned so that the operator may be used in a statement containing other operations.
- *date<sub>in</sub>* is the date to be added to the ZIL\_DATE object.

## Example

```
#include <ui_gen.hpp>

AddOneWeek(ZIL_DATE currentDate, ZIL_DATE &nextWeek)
{
    ZIL_INT32 oneWeek = 7;

    // Adding 1 week to the current date gives the next week.
    nextWeek = currentDate + oneWeek;
}
```

## ZIL\_DATE::operator -

### Syntax

```
#include <ui_gen.hpp>

ZIL_INT32 operator - (ZIL_INT32 days);
    or
ZIL_INT32 operator - (const ZIL_DATE &date);
```

### Portability

This function is available on the following environments:

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

## Remarks

The first operator overload subtracts the value *days* from the ZIL\_DATE object.

- *returnValue<sub>out</sub>* is the number of days resulting from the subtraction operation. This raw value is returned so that the operator may be used in a statement containing other operations.
- *days<sub>in</sub>* is the number of days to be subtracted from the ZIL\_DATE object.

The second operator overload subtracts the date contained in *date* from the ZIL\_DATE object.

- *returnValue<sub>out</sub>* is the difference, in days, between the ZIL\_DATE object and the date contained in *date*. This raw value is returned so that the operator may be used in a statement containing other operations.
- *date<sub>in</sub>* is the date to be subtracted from the ZIL\_DATE object.

## Example

```
#include <ui_gen.hpp>

SubtractOneWeek(ZIL_DATE currentDate, ZIL_DATE &lastWeek)
{
    ZIL_INT32 oneWeek = 7;

    // Subtracting 1 week from the current date gives the previous week.
    lastWeek = currentDate - oneWeek;
}
```

## ZIL\_DATE::operator >

### Syntax

```
#include <ui_gen.hpp>

int operator > (const ZIL_DATE &date);
```



## Portability

This function is available on the following environments:

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

## Remarks

This operator overload determines whether the `ZIL_DATE` object is chronologically greater than the date specified by *date*.

- *returnValue<sub>out</sub>* is TRUE if the `ZIL_DATE` object is chronologically greater than *date*. Otherwise, *returnValue* is FALSE.
- *date<sub>in</sub>* is the date to be compared.

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_DATE currentDate; // Initialize a system date.
    ZIL_DATE twentyFirstCentury("Jan. 1, 2000");

    // Check the dates.
    if (currentDate > twentyFirstCentury I I
        currentDate == twentyFirstCentury)
        printf("The twenty first century has already come.\n");
}
```

## ZIL\_DATE::operator >=

### Syntax

```
#include <ui_gen.hpp>

int operator >= (const ZIL_DATE &idate);
```

## Portability

This function is available on the following environments:

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

## Remarks

This operator overload determines whether the ZIL\_DATE object is chronologically greater than or equal to the date specified by *date*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_DATE object is chronologically greater than or equal to *date*. Otherwise, *returnValue* is FALSE.
- *date<sub>in</sub>* is the date to be compared.

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_DATE currentDate; // Initialize a system date.
    ZIL_DATE twentyFirstCentury("Jan. 1, 2000");

    // Check the dates.
    if (currentDate >= twentyFirstCentury)
        printf("The twenty first century has already come.\n");
}
```

## ZIL\_DATE::operator <

### Syntax

```
#include <ui_gen.hpp>

int operator < (const ZIL_DATE &date);
```

## Portability

This function is available on the following environments:

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

## Remarks

This operator overload determines whether the `ZIL_DATE` object is chronologically less than the date specified by *date*.

- *returnValue<sub>out</sub>* is TRUE if the `ZIL_DATE` object is chronologically less than *date*. Otherwise, *returnValue* is FALSE.
- *date<sub>in</sub>* is the date to be compared.

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_DATE currentDate; // Initialize a system date.
    ZIL_DATE twentyFirstCentury("Jan. 1, 2000");

    // Check the dates.
    if (currentDate < twentyFirstCentury)
        printf("It's not the twenty first century.\n");
}
```

## ZIL\_DATE::operator <=

### Syntax

```
#include <ui_gen.hpp>

int operator <= (const ZIL_DATE &date);
```

## Portability

This function is available on the following environments:

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

## Remarks

This operator overload determines whether the ZIL\_DATE object is chronologically less than or equal to the date specified by *date*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_DATE object is chronologically less than or equal to *date*. Otherwise, *returnValue* is FALSE.
- *date<sub>in</sub>* is the date to be compared.

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_DATE currentDate; // Initialize a system date.
    ZIL_DATE endOfTwentiethCentury("Dec. 31, 1999");

    // Check the dates.
    if (currentDate <= endOfTwentiethCentury)
        printf("It's not the twenty first century.\n");
}
```

## ZIL\_DATE-operator ++

### Syntax

```
#include <ui_gen.hpp>

ZIL_INT32 operator ++ (void);
```

## Portability

This function is available on the following environments:

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

## Remarks

This operator overload increments the value of the ZIL\_DATE object by one day.

- *returnValue<sub>out</sub>* is the number of days after the ZIL\_DATE object has been incremented. This raw value is returned so that the operator may be used in a statement containing other operations.

## Example

```
#include <ui_gen.hpp>
AdvanceCurrentDate(ZIL_DATE &currentDate)
{
    // Advance the current date.
    ++currentDate;
}
```

## ZIL\_DATE::operator --

### Syntax

```
#include <ui_gen.hpp>

ZIL_INT32 operator -- (void);
```

## Portability

This function is available on the following environments:

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

## Remarks

This operator overload decrements the value of the ZIL\_DATE object by one day.

- *returnValue<sub>out</sub>* is the number of days after the ZIL\_DATE object has been decremented. This raw value is returned so that the operator may be used in a statement containing other operations.

## Example

```
#include <ui_gen.hpp>
DecrementCurrentDate(ZIL_DATE kcurrentDate)
{
    // Decrement the current date.
    --currentDate;
}
```

## ZIL\_DATE::operator +=

### Syntax

```
#include <ui_gen.hpp>

void operator += (ZIL_INT32 days);
```

### Portability

This function is available on the following environments:

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

## Remarks

This operator overload adds *days* to the ZIL\_DATE object and copies the result back into the ZIL\_DATE object.

- *days<sub>in</sub>* is the number of days to be added to the ZIL\_DATE object.

## Example

```
#include <ui_gen.hpp>

AddOneWeek(ZIL_DATE currentDate, ZIL_DATE &nextWeek)
{
    ZIL_INT32 oneWeek = 7;

    // Adding 1 week to the current date gives the next week.
    nextWeek = currentDate;
    nextWeek += oneWeek;
}
```

## ZIL\_DATE::operator -=

### Syntax

```
#include <ui_gen.hpp>

void operator -= (ZIL_INT32 days);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload subtracts *days* from the ZIL\_DATE object and copies the result back into the ZIL\_DATE object.

- *days<sub>in</sub>* is the number of days to be subtracted from the ZIL\_DATE object.

## Example

```
#include <ui_gen.hpp>
SubtractWeeks(ZIL_DATE currentDate, ZIL_DATE &lastWeek)
{
    ZIL_INT32 oneWeek = 7;

    // Subtracting 1 week from the current date gives the previous week.
    lastWeek = currentDate;
    lastWeek -= oneWeek;
}
```

## ZIL\_DATE::operator ==

### Syntax

```
#include <ui_gen.hpp>

int operator == (const ZIL_DATE &date);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload determines whether the ZIL\_DATE object is chronologically equal to the date specified by *date*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_DATE object is chronologically equal to *date*. Otherwise, *returnValue* is FALSE.
- *date<sub>in</sub>* is the date to be compared.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_DATE currentDate; // Initialize a system date.
    ZIL_DATE newYears1990 ("Jan. 1, 1990");

    // Check the dates,
    if (currentDate == newYears1990)
        printf("It's new years day 1990.\n")
}
}
```



## ZIL\_DATE::operator !=

### Syntax

```
#include <ui_gen.hpp>

int operator != (const ZIL_DATE &date);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload determines whether the ZIL\_DATE object is chronologically not equal to the date specified by *date*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_DATE object is chronologically not equal to *date*. Otherwise, *returnValue* is FALSE.
- *date<sub>in</sub>* is the date to be compared.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_DATE currentDate; // Initialize a system date.
    ZIL_DATE newYears1990("Jan. 1, 1990");

    // Check the dates,
    if (currentDate != newYears1990)
        printf("It is not new years day 1990.\n")
}
}
```

## CHAPTER 52 - ZIL DECORATION

The ZIL\_DECORATION class object is used to maintain decorations, or images, for an object. Any object that needs to be drawn by OpenZinc has a pointer to the ZIL\_DECORATION object containing that object's images. The object can get a pointer to the appropriate ZIL\_DECORATION object through the ZIL\_DECORATION\_MANAGER, which maintains a list of all ZIL\_DECORATION objects. The images are kept in ZIL\_TEXT\_ELEMENT and ZIL\_BITMAP\_ELEMENT objects. Because each instance of an object can have its own ZIL\_DECORATION object, any combination of locale images can be used simultaneously.

The ZIL\_DECORATION class is declared in UI\_GEN.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS ZIL_DECORATION : public ZIL_I18N
{
public:
    ZIL_DECORATION(void);
    #if defined(ZIL_LOAD)
    virtual void ClassLoadData(ZIL_STORAGE_OBJECT_READ_ONLY *object);
    #endif
    #if defined(ZIL_STORE)
    virtual void ClassStoreData(ZIL_STORAGE_OBJECT *object);
    #endif
    int noOfTextElements;
    ZIL_TEXT_ELEMENT *text;
    int noOfBitmapElements;
    ZIL_BITMAP_ELEMENT *bitmap;
    ZIL_ICHAR *GetText(ZIL_NUMBERID numberID, int useDefault = FALSE) const;
    ZIL_UINT8 *GetBitmap(ZIL_NUMBERID numberID,
        int useDefault = FALSE) const;
protected:
    virtual void AssignData(const ZIL_I18N *data);
    virtual void DeleteData(void);
};
```

### General Members

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

- *noOfTextElements* indicates how many text images are maintained by the ZIL\_DECORATION object.
- *text* is the list of ZIL\_TEXT\_ELEMENT objects that contain the text images.
- *noOfBitmapElements* indicates how many bitmap images are maintained by the ZIL\_DECORATION object.

- *bitmap* is the list of ZIL\_BITMAP\_ELEMENT objects that contain the bitmap images.

## **ZIL\_DECORATION::ZIL\_DECORATION**

### **Syntax**

```
#include <ui_gen.hpp>

ZIL_DECORATION(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 ZIL\_DECORATION object.

## **ZIL\_DECORATION::AssignData**

### **Syntax**

```
#include <ui_gen.hpp>

virtual void AssignData(const ZIL_I18N *data);
```

### **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 the data maintained by *data* to the ZIL\_DECORATION object. The *noOfTextElements* value, the *noOfBitmapElements* value, the *text* pointer, and the *bitmap* pointer are copied. This function does not create a new copy of the data, but simply assigns the pointers.

- *data<sub>in</sub>* is a pointer to the ZIL\_DECORATION object containing the data that is to be assigned.

## ZIL\_DECORATION::DeleteData

### Syntax

```
#include <ui_gen.hpp>

virtual void DeleteData(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 deletes the data maintained by this object if *allocated* is TRUE. The text in each ZIL\_TEXT\_ELEMENT, the bitmap in each ZIL\_BITMAP\_ELEMENT, the list of ZIL\_TEXT\_ELEMENT objects, and the list of ZIL\_BITMAP\_ELEMENT objects are deleted.

## ZIL\_DECORATION::GetBitmap

### Syntax

```
#include <ui_gen.hpp>

ZIL_UINT8 *GetBitmap(ZIL_NUMBERID numberID, int useDefault = FALSE) const;
```

## 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 bitmap maintained by the `ZIL_BITMAP_ELEMENT` object identified by *numberID*.

- *returnValue<sub>out</sub>* is a pointer to the bitmap.
- *numberID<sub>in</sub>* is a value identifying the `ZIL_BITMAP_ELEMENT`.
- *useDefault<sub>in</sub>* indicates if the default bitmap should be used if no match is found on *numberID*. If *useDefault* is `TRUE`, the bitmap from the first `ZIL_BITMAP_ELEMENT` is returned if no `ZIL_BITMAP_ELEMENT` objects matched *numberID*. If no match was found and *useDefaults* is `FALSE`, `NULL` is returned.

## ZIL\_DECORATION::GetText

### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_ICHAR *GetText(ZIL_NUMBERID numberID, int useDefault = FALSE) const;
```

## 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 image maintained by the `ZIL_TEXT_ELEMENT` object identified by *numberID*.

- *returnValue<sub>out</sub>* is a pointer to the text string.
- *numberID<sub>in</sub>* is a value identifying the ZIL\_TEXT\_ELEMENT.
- *useDefault<sub>in</sub>* indicates if the default text image should be used if no match is found on *numberID*. If *useDefault* is TRUE, the text from the first ZIL\_TEXT\_ELEMENT is returned if no ZIL\_TEXT\_ELEMENT objects matched *numberID*. If no match was found and *useDefaults* is FALSE, NULL is returned.

## Storage Members

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

### ZIL\_DECORATION::ClassLoadData

#### Syntax

```
#include <ui_gen.hpp>
```

```
virtual void ClassLoadData(ZIL_STORAGE_OBJECT_READ_ONLY *object);
```

#### 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 ZIL\_DECORATION data from a persistent object data file. The data is loaded from the current directory. This function is typically not used by the programmer.

- *object<sub>in</sub>* is a pointer to the ZIL\_STORAGE\_OBJECT\_READ\_ONLY object that contains the data. For more information on persistent objects, see "Chapter 69—ZIL\_STORAGE\_OBJECT\_READ\_ONLY."

## ZIL DECORATION::ClassStoreData

### Syntax

```
#include <ui_gen.hpp>
```

```
virtual void ClassStoreData(ZIL_STORAGE_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 advanced function is used to store ZIL\_DECORATION data in a persistent object data file. The data is stored in the current directory. This function is typically not used by the programmer.

- *object<sub>in</sub>* is a pointer to the ZIL\_STORAGE\_OBJECT where the data will be stored. For more information on persistent objects, see "Chapter 68—ZIL\_STORAGE\_OBJECT."

## CHAPTER 53 - ZIL\_DECORATION\_MANAGER

The ZIL\_DECORATION\_MANAGER class object is used to maintain a list of ZIL\_DECORATION objects. Each ZIL\_DECORATION object contains decorations for library objects. A decoration is the image used to draw the object. In graphics mode, the decoration is a bitmap that is displayed on the object. In text mode, the decoration is a text string made up of the characters used to display the object. Because OpenZinc does all the drawing in DOS and Curses, this class is used extensively in these environments. Other environments may or may not use it for some objects.

The ZIL\_DECORATION\_MANAGER class is declared in **UI\_GEN.HPP**. Its public and protected members are:

```
class ZIL_EXPORT_CLASS ZIL_DECORATION_MANAGER : public ZIL_I18N_MANAGER
public:
    ZIL_DECORATION_MANAGER(void);
    virtual ZIL_I18N *CreateData(void);

    static void FreeDecorations(const ZIL_DECORATION *decorations);
    static void LoadDefaultDecorations(const ZIL_ICHAR *decorationName);
    static const ZIL_DECORATION *UseDecorations(const ZIL_DECORATION
        *decorations);
    static const ZIL_DECORATION *UseDecorations(const ZIL_ICHAR *className,
        const ZIL_ICHAR *decorationName = ZIL_NULLP(ZIL_ICHAR));

    static void SetDecorations(const ZIL_ICHAR *className,
        ZIL_TEXT_ELEMENT *defaultText, ZIL_BITMAP_ELEMENT *defaultBitmap);
};
```

### General Members

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

### ZIL\_DECORATION\_MANAGER::ZIL\_DECORATION\_MANAGER

#### Syntax

```
#include <ui_gen.hpp>

ZIL_DECORATION_MANAGER(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 ZIL\_DECORATION\_MANAGER object.

## ZIL\_DECORATION\_MANAGER::CreateData

### Syntax

```
#include <ui_gen.hpp>
```

```
virtual ZIL_I18N *CreateData(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 creates a new ZIL\_DECORATION object. Because it is a pure virtual function at the base ZIL\_I18N\_MANAGER class level, the generic code in the ZIL\_I18N\_MANAGER class can use it to create the proper data object.

- *returnValue<sub>out</sub>* is a pointer to the new ZIL\_DECORATION object that was created.

## ZILDECORATION\_MANAGER::FreeDecorations

### Syntax

```
#include <ui_gen.hpp>

static void FreeDecorations(const ZIL_DECORATION *decorations);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function releases the ZIL\_DECORATION object from use by decrementing the ZIL\_DECORATION object's *useCount* member. Whenever a library object requests the use of a ZIL\_DECORATION object, it does so by calling the **UseDecorations( )** function, which marks the ZIL\_DECORATION object as used by incrementing its *useCount* member. When the library object is done using the ZIL\_DECORATION object, it must release it by calling this function. If the releasing object was the last object using the ZIL\_DECORATION object, this function will deallocate the data being maintained by the ZIL\_DECORATION object unless it contains the default data.

- *decorations<sub>in</sub>* is a pointer to the ZIL\_DECORATION object that is being released.

## ZIL\_DECORATION\_MANAGER::LoadDefaultDecorations

### Syntax

```
#include <ui_gen.hpp>

static void LoadDefaultDecorations(const ZIL_ICHAR *decorationsName)\
```

## 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 default decorations for the application. Any objects that are using the default decorations at the time this function is called will start using the new default images. If necessary, this function loads the default images data from the **I18N.DAT** file.

- *decorationsName<sub>in</sub>* is the two-letter ISO country name identifying which images are to be the default for the application.

## ZIL\_DECORATION\_MANAGER::SetDecorations

### Syntax

```
#include <ui_gen.hpp>
```

```
static void SetDecorations(const ZIL_ICHAR *className,  
                           ZIL_TEXT_ELEMENT *defaultText, ZIL_BITMAP_ELEMENT *defaultBitmap);
```

## 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 default images for a particular object in the list of images maintained by the ZIL\_DECORATION\_MANAGER class. An image must be placed in the list using this function before it can be accessed using the **UseDecorations( )** function. The images are assumed to be for the country identified by the *isoImageName* global variable. This variable and the default images are defined in the **IMG\_DEF.CPP** file. If different default images are desired, simply copy an **IMG\_<ISO>.CPP** file from the

ZINC\SOURCE\INTL directory to the ZINC\SOURCE directory and rename it to **IMG\_DEF.CPP**. Then rebuild the library.

- *className<sub>in</sub>* is the class name of the object for which the image is being set. This typically corresponds to the *jclassName* member variable of the object.
- *defaultText<sub>in</sub>* is the ZIL\_TEXT\_ELEMENT class that contains the text mode character image for the object.
- *defaultBitmap<sub>in</sub>* is the ZIL\_BITMAP\_ELEMENT class that contains the graphics mode bitmap image for the object.

## ZIL\_DECORATION\_MANAGER::UseDecorations

### Syntax

```
#include <ui_gen.hpp>

static ZIL_DECORATION *UseDecorations(const ZIL_DECORATION *decorations);
    or
static const ZIL_DECORATION *UseDecorations(const ZIL_ICHAR
    *decorationsName = 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

These overloaded functions mark a ZIL\_IDECORATION object as used by incrementing its *useCount* member. Whenever a library object requests the use of a ZIL\_DECORATION object, it marks the object as used by calling this function, which increments the object's *useCount* member. When the library object is done using the ZIL\_DECORATION object, it must release it by calling the **FreeDecorations()** function.

The first overloaded function takes a pointer to the ZIL\_DECORATION object being marked as used.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_DECORATION object.
- *language<sub>in</sub>* is a pointer to the ZIL\_DECORATION object that is to be marked as used.

The second overloaded function takes the decorations name. If load capability is enabled (i.e., **ZIL\_LOAD** was defined when the library was compiled) this function will load the data from the **I18N.DAT** file if necessary. Otherwise, the data must have been compiled and linked into the application.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_DECORATION object.
- *decorationsName<sub>in</sub>* is the two-letter ISO country name identifying which images are to be used.

## CHAPTER 54 - ZIL\_DELTA\_STORAGE\_OBJECT

The ZIL\_DELTA\_STORAGE\_OBJECT class object is used to store changes, or deltas, to objects. By using delta storage, entire objects don't need to be saved if only a small part of the object changed. A prime use for delta storage is for creating international applications. A window with several dozen objects on it may be created in the Designer. If the application is to be used in several different languages, then the strings on the window need to be translated. Rather than storing a complete copy of the window for each language, however, the original window is stored and only the changes to the objects, such as the new text, are saved.

The ZIL\_DELTA\_STORAGE\_OBJECT class is declared in UI\_GEN.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS ZIL_DELTA_STORAGE_OBJECT : public ZIL_STORAGE_OBJECT
{
public:
    // Read/Write support
    ZIL_DELTA_STORAGE_OBJECT(ZIL_STORAGE_OBJECT_READ_ONLY *object,
        ZIL_STORAGE &file, const ZIL_ICHAR *name,
        ZIL_OBJECTID nObjectID, UIS_FLAGS pFlags = UIS_READWRITE);
    ~ZIL_DELTA_STORAGE_OBJECT(void);
    int Store(ZIL_INT16 value);
    int Store(ZIL_UINT16 value);
    int Store(ZIL_INT32 value);
    int Store(ZIL_UINT32 value);
    int Store(ZIL_INT8 value);
    int Store(ZIL_UINT8 value);
    int Store(void *buff, int size, int length);
    int Store(const ZIL_ICHAR *string);
};
```

### General Members

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

### ZIL\_DELTA\_STORAGE\_OBJECT::ZIL\_DELTA\_STORAGE\_OBJECT

#### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_DELTA_STORAGE_OBJECT(ZIL_STORAGE_OBJECT_READ_ONLY *object,
    ZIL_STORAGE &file, const ZIL_ICHAR *name,
```

ZIL\_OBJECTID *nObjectID*, UIS\_FLAGS *pFlags* = UIS\_READWRITE);

## 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 ZIL\_DELTA\_STORAGE\_OBJECT class object.

- *object<sub>in</sub>* is the pointer to the original object. When the new object is stored, this pointer is referenced to determine if a delta version of the object needs to be stored.
- *file<sub>in</sub>* is the file where the delta object will be stored.
- *name<sub>in</sub>* is the name of the delta object.
- *nObjectID<sub>in</sub>* is the *objectID* for the delta object.
- *pFlags<sub>in</sub>* indicates how the storage object is to be opened. The following UIS\_FLAGS are supported:

**UIS\_READ**—Allows read only access to the object.

**UIS\_READWRITE**—Allows read and write access to the object. This flag allows modifications to be made to the object.

**UIS\_CREATE**—Creates an object and allows write access to it. Any previous object will be deleted.

**UIS\_OPENCREATE**—Opens an existing object for read and write access. If the object does not exist, it is created for read and write access.

## **ZIL\_DELTA\_STORAGE\_OBJECT::~~ZIL\_DELTA\_STORAGE\_OBJECT**

### **Syntax**

```
#include <ui_gen.hpp>
```

```
virtual ~ZIL_DELTA_STORAGE_OBJECT(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 ZIL\_DELTA\_STORAGE\_OBJECT object.

## **ZIL\_DELTA\_STORAGE\_OBJECT::Store**

### **Syntax**

```
#include <ui_gen.hpp>
```

```
int Store(ZIL_INT16 value);
```

*or*

```
int Store(ZIL_UINT16 value);
```

*or*

```
int Store(ZIL_INT32 value);
```

*or*

```
int Store(ZIL_UINT32 value);
```

*or*

```
int Store(ZIL_UINT8 value);
```

*or*

```
int Store(ZIL_INT8 value);
```

*or*

```
int Store(void* buff, int size, int length);
```



*or*  
int Store(const ZIL\_ICHAR \*string);

## Portability

These functions are available on the following environments:

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

## Remarks

The first six overloaded functions write information to the storage file according to the type of value given.

- *returnValue<sub>out</sub>* is the number of bytes written.
- *value<sub>in</sub>* is the numeric value to be written. The following values are supported:

**ZIL\_INT8**—A number whose value is between -128 and 127 (8 bits, signed).

**ZIL\_UINT8**—A number whose value is between 0 and 255 (8 bits, unsigned).

**ZIL\_INT16**—A number whose value is between -32,768 and 32,767 (16 bits, signed).

**ZIL\_IJINT16**—A number whose value is between 0 and 65,535 (16 bits, unsigned).

**ZIL\_INT32**—A number whose value is between -2,147,483,648 and 2,147,483,647 (32 bits, signed).

**ZIL\_UINT32**—A number whose value is between 0 and 4,294,967,295 (32 bits, unsigned).

The seventh overloaded function writes information into the storage file according to the following values:

- *returnValue<sub>out</sub>* is the number of bytes written.
- *buff<sub>in</sub>* is a pointer to the buffer that contains the information to be written.

- $size_{in}$  is the size of each item to be written.
- $length_{in}$  is the number of items to be written.

In general, programmers are discouraged from using this function, because the integrity of the type of value being stored cannot be guaranteed across environments. For example, the storage size of a value (e.g., int) in DOS might be different than that in Motif. All of the other **Store**( ) functions, however, are the same across environments.

The eighth overloaded function writes information into the storage file according to the following value:

- $returnValue_{out}$  is the number of bytes written.
- $string_{in}$  is a pointer to the string that is to be written.



## CHAPTER 55 - ZIL\_DELTA\_STORAGE\_OBJECT\_READ\_ONLY

The `ZIL_DELTA_STORAGE_OBJECT_READ_ONLY` class object is used to read changes, or deltas, for an object from a file. By using delta storage, entire objects don't need to be saved if only a small part of the object changed. A prime use for delta storage is for creating international applications. A window with several dozen objects on it may be created in the Designer. If the application is to be used in several different languages, then the strings on the window need to be translated. Rather than storing a complete copy of the window for each language, however, the original window is stored and only the changes to the objects, such as the new text, are saved.

The `ZIL_DELTA_STORAGE_OBJECT_READ_ONLY` class is declared in `UI_GEN.HPP`. Its public and protected members are:

```
class ZIL_EXPORT CLASS ZIL_DELTA_STORAGE_OBJECT_READ_ONLY :
public ZIL_STORAGE_OBJECT_READ_ONLY
{
public:
    // Read-Only support
    ZIL_DELTA_STORAGE_OBJECT_READ_ONLY(
        ZIL_STORAGE_OBJECT_READ_ONLY *object,
        ZIL_STORAGE_READ_ONLY &file, const ZIL_ICHAR *name,
        ZIL_OBJECTID nObjectID);
    ~ZIL_DELTA_STORAGE_OBJECT_READ_ONLY(void);

    int Load(ZIL_INT16 *value);
    int Load(ZIL_UINT16 *value);
    int Load(ZIL_INT32 *value);
    int Load(ZIL_UINT32 *value);
    int Load(ZIL_UINT8 *value);
    int Load(ZIL_INT8 *value);
    int Load(void *buff, int size, int length);
    int Load(ZIL_ICHAR *string, int length);
    int Load(ZIL_ICHAR **string);
```

### General Members

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

## **ZIL\_DELTA\_STORAGE\_OBJECT\_READ\_ONLY::ZIL\_DELTA\_STORAGE\_OBJECT\_READ\_ONLY**

### **Syntax**

```
#include <ui_gen.hpp>
```

```
ZIL_DELTA_STORAGE_OBJECT_READ_ONLY(  
    ZIL_STORAGE_OBJECT_READ_ONLY *object,  
    ZIL_STORAGE_READ_ONLY &file, const ZIL_ICHAR *name,  
    ZIL_OBJECTID nObjectID);
```

### **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 `ZIL_DELTA_STORAGE_OBJECT_READ_ONLY` class object. The `ZIL_DELTA_STORAGE_OBJECT_READ_ONLY` that is created is passed to the `Load()` member function of the window object as the `ZIL_STORAGE_OBJECT_READ_ONLY` parameter.

- *object<sub>in</sub>* is a pointer to the object. If a pointer to the object that is already opened is passed in, both the original object and the deltas are loaded. If this pointer is `NULL`, it is assumed that the original object has already been loaded and only the deltas are loaded.
- *file<sub>in</sub>* is the file where the delta object is located.
- *name<sub>in</sub>* is the name of the delta object.
- *nObjectID<sub>in</sub>* is the *objectID* for the delta object.

## **ZIL\_DELTA\_STORAGE\_OBJECT\_READ\_ONLY::~~ZIL\_DELTA\_STORAGE\_OBJECT\_READ\_ONLY**

### **Syntax**

```
#include <ui_gen.hpp>
```

```
virtual ~ZIL_DELTA_STORAGE_OBJECT_READ_ONLY(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 `ZIL_DELTA_STORAGE_OBJECT_READ_ONLY` object.

## **ZIL\_DELTA\_STORAGE\_OBJECT\_READ\_ONLY::Load**

### **Syntax**

```
#include <ui_gen.hpp>
```

```
int Load(ZIL_INT16 *value);
```

*or*

```
int Load(ZIL_UINT16 *value);
```

*or*

```
int Load(ZIL_INT32 *value);
```

*or*

```
int Load(ZIL_UINT32 *value);
```

*or*

```
int Load(ZIL_UINT8 *value);
```

*or*

```
int Load(ZIL_INT8 *value);
```

*or*

```
int Load(void *buff, int size, int length);
```

*or*  
int Load(ZIL\_ICHAR \*string, int length);  
*or*  
int Load(ZIL\_ICHAR \*\*string);

## Portability

These functions are available on the following environments:

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

## Remarks

The first six overloaded functions read information from the storage file according to the type of value given.

- *returnValue<sub>out</sub>* is the number of bytes read.
- *value<sub>out</sub>* is the numeric value read. The following values are supported:

**ZIL\_INT8**—A number whose value is between -128 and 127 (8 bits, signed).

**ZIL\_UINT8**—A number whose value is between 0 and 255 (8 bits, unsigned).

**ZIL\_INT16**—A number whose value is between -32,768 and 32,767 (16 bits, signed).

**ZIL\_UINT16**—A number whose value is between 0 and 65,535 (16 bits, unsigned).

**ZIL\_INT32**—A number whose value is between -2,147,483,648 and 2,147,483,647 (32 bits, signed).

**ZIL\_UINT32**—A number whose value is between 0 and 4,294,967,295 (32 bits, unsigned).

The seventh overloaded function reads information from the storage file according to the following values:

*buff<sub>out</sub>* is a pointer to the buffer that will receive the information. This buffer must be large enough to contain the information read.

- *size<sub>in</sub>* is the size of each item to be read.
- *length<sub>in</sub>* is the number of items to be read.

In general, programmers are discouraged from using this function, because the integrity of the type of value being loaded cannot be guaranteed across environments. For example, the storage size of a value in DOS might be different than that in Motif. All of the other **Load()** functions, however, are the same across environments.

The eighth overloaded function reads information from the storage file according to the following values:

- *string<sub>in</sub>* is a pointer to the character buffer that will receive the information. This buffer must be large enough to contain the information read.
- *length<sub>in</sub>* is the number of characters to read.

The ninth overloaded function reads information from the storage file according to the following values:

- *string<sub>out</sub>* is a pointer to a string pointer where the information will be written. This string is allocated by the library.





## CHAPTER 56 - ZIL\_118N

The ZIL\_118N class is the base class for the classes that maintain internationalization data. Derived classes include ZIL\_DECORATION, ZIL\_LANGUAGE, and ZIL\_LOCALE. The ZIL\_118N class provides those member variables and functions that are common to the derived classes.

The ZIL\_118N class is declared in **UI\_GEN.HPP**. Its public and protected members are:

```
Class ZIL_EXPORT_CLASS ZIL_118N : public UI_ELEMENT
{
    friend class ZIL_EXPORT_CLASS ZIL_118N_MANAGER;
public:
    const ZIL_ICHAR *className;
    const ZIL_ICHAR *pathName;
    ZIL_ICHAR name[12];
    int useCount;
    int error;

    ZIL_118N(void);
    ~ZIL_118N(void);

#if defined(ZIL_LOAD)
    static int Traverse(ZIL_STORAGE_READ_ONLY *storage,
        const ZIL_ICHAR *path);
    void Load(ZIL_STORAGE_READ_ONLY *storage,
        ZIL_STORAGE_OBJECT_READ_ONLY *object);
    virtual void ClassLoadData(ZIL_STORAGE_OBJECT_READ_ONLY *object);
#endif
#if defined(ZIL_STORE)
    static int Traverse(ZIL_STORAGE *storage, const ZIL_ICHAR *path,
        int create = FALSE);
    void Store(ZIL_STORAGE *storage, ZIL_STORAGE_OBJECT *object);
    virtual void ClassStoreData(ZIL_STORAGE_OBJECT *object);
#endif
    static ZIL_STORAGE_READ_ONLY *defaultStorage;
    static ZIL_ICHAR *il8nName;
protected:
    ZIL_UINT8 allocated;
    ZIL_UINT8 defaults;

    virtual void AssignData(const ZIL_118N *data);
    virtual void DeleteData(void);
};
```

### General Members

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

- *className* is a string that identifies the object with which the data maintained by the ZIL\_118N class is associated.

- *pathName* is the pathname off the ZIL\_INTERNATIONAL directory within the **I18N.DAT** internationalization data file where the data for this object is located. *pathName* is "DECORATION" for the ZIL\_DECORATION class, "LANGUAGE" for the ZIL\_LANGUAGE class, and "LOCALE" for the ZIL\_ILOCALE class.
- *name* identifies the country or language for which the internationalization data applies. Typically this is the two-letter ISO country or language code.
- *useCount* indicates how many objects are currently using the instance of this object. *useCount* is updated in the **UseI18N()** and **FreeI18N()** functions and their derived equivalents (i.e., **UseLanguage()**, etc.).
- *error* contains any error codes returned by *defaultStorage*.
- *defaultStorage* is the data file that contains internationalization data. By default this file is **I18N.DAT**.
- *il8nName* is the name of the internationalization data file. By default, *il8nName* is "i18n.dat."
- *allocated* indicates if the memory for the data maintained by the object was allocated by the library. If the memory was allocated by the library, *allocated* is TRUE. Otherwise, *allocated* is FALSE. If *allocated* is TRUE, the data can be deleted by the library.
- *defaults* indicates if the instance of this class contains the default data.

## ZIL I18N::ZIL I18N

### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_I18N(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 ZIL\_I18N class object.

## ZIL\_I18N::~ZIL\_I18N

### Syntax

```
#include <ui_gen.hpp>
```

```
~ZIL_I18N(void);
```

### Portability

This function is available on the following environments:

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

## Remarks

This destructor destroys the information associated with the ZIL\_I18N object, including the data maintained by the class if *allocated* is TRUE.

## ZIL\_I18N::AssignData

### Syntax

```
#include <ui_gen.hpp>
```

```
virtual void AssignData(const ZIL_I18N *data)\
```

### 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 a stub that does nothing. It is provided in the event that a derived class does not need to implement the function.

- *data<sub>m</sub>* is not used.

## **ZIL I18N::DeleteData**

### Syntax

```
#include <ui_gen.hpp>

virtual void DeleteData(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 a stub that does nothing. It is provided in the event that a derived class does not need to implement the function.

## Storage Members

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

## ZIL\_I18N::ClassLoadData

### Syntax

```
#include <ui_gen.hpp>
```

```
virtual void ClassLoadData(ZIL_STORAGE_OBJECT_READ_ONLY *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 a stub that does nothing. It is provided in the event that a derived class does not need to implement the function.

- *object<sub>in</sub>* is not used.

## ZIL\_I18N::ClassStorePata

### Syntax

```
#include <ui_gen.hpp>
```

```
virtual void ClassStoreData(ZIL_STORAGE_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 a stub that does nothing. It is provided in the event that a derived class does not need to implement the function.

- *object<sub>in</sub>* is not used.

## ZIL M8N::Load

### Syntax

```
#include <ui_gen.hpp>

virtual void Load(ZIL_STORAGE_READ_ONLY *storage,
                 ZIL_STORAGE_OBJECT_READ_ONLY *object);
```

### 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 ZIL\_I18N data from a persistent object data file. This function is typically not used by the programmer.

- *storage<sub>in</sub>* is a pointer to the ZIL\_STORAGE\_READ\_ONLY object that contains the data. For more information on persistent object files, see "Chapter 70—ZIL\_STORAGE\_READ\_ONLY."
- *object<sub>in</sub>* is a pointer to the ZIL\_STORAGE\_OBJECT\_READ\_ONLY where the data will be loaded. For more information on loading information from persistent object files, see "Chapter 69—ZIL\_STORAGE\_OBJECT\_READ\_ONLY."

## ZIL\_I18N::Store

### Syntax

```
#include <ui_gen.hpp>
```

```
virtual void Store(ZIL_STORAGE *storage, ZIL_STORAGE_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 advanced function is used to store ZIL\_I18N data in a persistent object data file. This function is typically not used by the programmer.

- *storage<sub>m</sub>* is a pointer to the ZIL\_STORAGE where the data will be stored. For more information on persistent object files, see "Chapter 66—ZIL\_STORAGE."
- *object<sub>m</sub>* is a pointer to the ZIL\_STORAGE\_OBJECT where the data will be stored. For more information on loading persistent objects, see "Chapter 68—ZIL\_STORAGE\_OBJECT."

## ZIL\_I18N: Traverse

### Syntax

```
static int Traverse(ZIL_STORAGE_READ_ONLY *storage, const ZIL_ICHAR *_path);  
or  
static int Traverse(ZIL_STORAGE *storage, const ZIL_ICHAR *_path,  
int create = FALSE);
```



## Portability

This function is available on the following environments:

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

## Remarks

These functions traverse the internationalization data file to the directory specified by *path*. The first function is used for read-only files. The second function is used for files that can be written to.

- *returnValue<sub>out</sub>* indicates whether the function was able to traverse the internationalization data file to the appropriate subdirectory. If the function was able to traverse the data file correctly, *returnValue* will be 0. If the function was not successful, *returnValue* will be non-zero.
- *storage<sub>in</sub>* is a pointer to the data file where the internationalization data is located. Typically, this will be the data file pointed to by *defaultStorage*.
- *-path<sub>in</sub>* is the path to which the file should be traversed.
- *create<sub>in</sub>* specifies if the function should create the subdirectory if it does not already exist. If *create* is TRUE, the subdirectory will be created. Otherwise, the subdirectory will not be created.

## CHAPTER 57 - ZIL\_118N\_MANAGER

The ZIL\_118N\_MANAGER class object is an abstract class that defines the behavior of derived internationalization manager classes. The manager classes maintain a list of "data blocks" that contain bitmaps, text images, locale information, or language translations. Derived manager classes include the ZIL\_DECORATION\_MANAGER, the ZIL\_LANGUAGE\_MANAGER and the ZIL\_LOCALE\_MANAGER.

The ZIL\_118N\_MANAGER class is declared in UI\_GEN.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS ZIL_I18N_MANAGER : public UI_LIST
{
public:
    ZIL_ICHAR *defaultName;
    virtual ZIL_I18N *CreateData(void) = 0;
    void FreeI18N(const ZIL_I18N *il8n);
    void LoadDefaultI18N(const ZIL_ICHAR *il8nName);
    ZIL_I18N *UseI18N(const ZIL_I18N *il8n);
    ZIL_I18N *UseI18N(const ZIL_ICHAR *className,
                     const ZIL_ICHAR *il8nName);
};
```

### General Members

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

- *defaultName* is the two-letter ISO name identifying the default language or locale for the application.

### ZIL\_118N\_MANAGER::CreateData

#### Syntax

```
#include <ui_gen.hpp>
```

```
virtual ZIL_I18N *CreateData(void) = 0;
```

## 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 `ZIL_I18N` object (or derived object). This function is a pure virtual function and so it has no definition for the `ZIL_I18N_MANAGER` class. Each derived manager class implements this function.

- `returnValueout` is a pointer to the new `ZIL_I18N` object (or derived object) that was created.

## ZIL\_I18N\_MANAGER::FreeI18N

### Syntax

```
#include <ui_gen.hpp>
```

```
void FreeI18N(const ZIL_I18N *i18n);
```

## Portability

This function is available on the following environments:

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

## Remarks

This function releases the `ZIL_I18N` object from use by decrementing the `ZIL_I18N` object's `useCount` member. Whenever a library object requests the use of a `ZIL_I18N` object, it does so by calling the `UseI18N( )` function, which marks the `ZIL_I18N` object as used by incrementing its `useCount` member. When the library object is done using the `ZIL_I18N` object, it must release it by calling this function. If the releasing object was the last object using the `ZIL_I18N` object, this function will deallocate the data being maintained by the `ZIL_I18N` object unless it contains the default data.

- *i18n<sub>in</sub>* is a pointer to the ZIL\_I18N object that is being released.

## **ZIL\_I18N MANAGER::LoadDefaultI18N**

### **Syntax**

```
#include <ui_gen.hpp>
```

```
void LoadDefaultI18N(const ZIL_ICHAR *i18nName);
```

### **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 default language or locale for the application. Any objects that are using the default information at the time this function is called will start using the new default data. If necessary, this function loads the default internationalization data from the I18N.DAT file.

- *i18nName<sub>in</sub>* is the two-letter ISO name identifying the language or locale which is to be the default for the application.

## **ZIL\_I18N MANAGER::UseI18N**

### **Syntax**

```
#include <ui_gen.hpp>
```

```
ZIL_I18N *UseI18N(const ZIL_I18N *i18n);
```

*or*

```
ZIL_I18N *UseI18N(const ZIL_ICHAR *className, const ZIL_ICHAR *i18nName)\
```

## 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 mark a `ZIL_I18N` object as used by incrementing its `useCount` member. Whenever a library object requests the use of a `ZIL_I18N` object, it marks the object as used by calling this function, which increments the object's `useCount` member. When the library object is done using the `ZIL_I18N` object, it must release it by calling the `FreeI18N()` function.

The first overloaded function takes a pointer to the `ZIL_I18N` object being marked as used.

- `returnValueout` is a pointer to the `ZIL_I18N` object.
- `i18nin` is a pointer to the `ZIL_I18N` object that is to be marked as used.

The second overloaded function takes the class name of the object and the internationalization name. If load capability is enabled (i.e., `ZIL_LOAD` was defined when the library was compiled) this function will load the data from the `I18N.DAT` file if necessary. Otherwise, the data must have been compiled and linked into the application.

- `returnValueout` is a pointer to the `ZIL_I18N` object.
- `classNamein` is the class name of the object for which the internationalization data is being requested. This typically corresponds to the `_className` member variable of the object.
- `i18nNamein` is the two-letter ISO name identifying which set of data is requested. `i18nName` is either the country code or the language code, depending on the type of derived manager making the request. In this way, the country- or language-specific data for an object can be used.

## CHAPTER 58 - ZIL INTERNATIONAL

The ZIL\_INTERNATIONAL class is the base class for internationalization in OpenZinc Application Framework. This class maintains the default ZIL\_LOCALE and ZIL\_MAP\_CHARS classes and also provides many replacement functions for **Ctype**, string and file functions. These functions need to be overloaded to support Unicode 16-bit characters.

All UIW\_ objects in the library are derived from ZIL\_INTERNATIONAL through UI\_WINDOW\_OBJECT, thus giving the objects access to these overloaded functions through inheritance. Because of this, accessing the overloaded functions from within a member function is transparent; by calling **strdup**, for instance, you get the OpenZinc-overloaded **strdup**. So this portable functionality comes with no extra effort to you, the programmer. In addition, all these overloaded functions are static members and public, so the functions can be accessed globally as well.

The ZIL\_INTERNATIONAL class structure is declared in **UI\_GEN.HPP**. Its public and protected members are:

```
Class ZIL_EXPORT_CLASS ZIL_INTERNATIONAL
{
    friend class ZIL_EXPORT_CLASS ZIL_LOCALE;
    friend class ZIL_EXPORT_CLASS ZIL_LANGUAGE;
    friend class ZIL_EXPORT_CLASS ZIL_MAP_CHARS;
public:
    friend class ZIL_EXPORT_CLASS ZIL_STORAGE;
    friend class ZIL_EXPORT_CLASS ZIL_STORAGE_OBJECT;

    virtual ~ZIL_INTERNATIONAL(void);

    // Posix time() support
    static ZIL_INT32 minutesWestGMT;
    void TimeStamp(ZIL_UINT32 *value);

    static void DefaultI18nInitialize(void);
    static void CharMapInitialize(void);
    static void OSI18nInitialize(ZIL_ICHAR *langName,
        int forcelnitialization = FALSE);
    static void I18nInitialize(const ZIL_ICHAR localeName,
        const ZIL_ICHAR languageName);

    // Support for international Unicode/ISO helper functions,
    static int chartod(const ZIL_ICHAR value);
    static int IsNonspacing(ZIL_ICHAR value);

    // Support for international Ctype functions.
    static int IsAlnum(ZIL_ICHAR value);
    static int IsAlpha(ZIL_ICHAR value);
    static int IsAscii(ZIL_ICHAR value);
    static int IsCntrl(ZIL_ICHAR value);
    static int IsDigit(ZIL_ICHAR value);
    static int IsGraph(ZIL_ICHAR value);
    static int IsLower(ZIL_ICHAR value);
    static int IsPrint(ZIL_ICHAR value);
    static int IsPunct(ZIL_ICHAR value);
    static int IsSpace(ZIL_ICHAR value);
    static int IsUpper(ZIL_ICHAR value);
};
```

```

static int IsXDigit(ZIL_ICHAR value);
static ZIL_ICHAR ToLower(ZIL_ICHAR value);
static ZIL_ICHAR ToUpper(ZIL_ICHAR value);

// Support for internationalized ANSI routines.
#if defined(ZIL_NEXTSTEP)
static double strtod(const ZIL_ICHAR *nptr);
static int strtol(const ZIL_ICHAR *nptr);
static long strtoul(const ZIL_ICHAR *nptr);
#else
static int atoi(const ZIL_ICHAR *nptr);
static long atol(const ZIL_ICHAR *nptr);
static double atof(const ZIL_ICHAR *nptr);
#endif

static long strtol(const ZIL_ICHAR *nptr, ZIL_ICHAR **endptr, int base);
static unsigned long strtoul(const ZIL_ICHAR *nptr, ZIL_ICHAR **endptr,
int base);
static double strtod(const ZIL_ICHAR *nptr, ZIL_ICHAR **endptr);

static ZIL_ICHAR *strcpy(ZIL_ICHAR *s1, const ZIL_ICHAR *s2);
static int strcmp(const ZIL_ICHAR *s1, const ZIL_ICHAR *s2);
static int strncmp(const ZIL_ICHAR *s1, const ZIL_ICHAR *s2, int n);
static ZIL_ICHAR *strncpy(ZIL_ICHAR *s1, const ZIL_ICHAR *s2, int n);

static ZIL_ICHAR *strcat(ZIL_ICHAR *s1, const ZIL_ICHAR *s2);
static ZIL_ICHAR *strncat(ZIL_ICHAR *s1, const ZIL_ICHAR *s2, int n);

#if !defined(__SC__) || defined(ZIL_MACINTOSH)
static int strcoll(const ZIL_ICHAR *s1, const ZIL_ICHAR *s2);
#endif
#endif

static int strxfrm(ZIL_ICHAR *s1, const ZIL_ICHAR *s2, int n);
static ZIL_ICHAR *strchr(const ZIL_ICHAR *s, int c);
static int strcspn(const ZIL_ICHAR *s1, const ZIL_ICHAR *s2);
static ZIL_ICHAR *strpbrk(const ZIL_ICHAR *s1, const ZIL_ICHAR *s2);
static ZIL_ICHAR *strchr(const ZIL_ICHAR *s, int c);
static int strspn(const ZIL_ICHAR *s1, const ZIL_ICHAR *s2);
static ZIL_ICHAR *strstr(const ZIL_ICHAR *s1, const ZIL_ICHAR *s2);
static ZIL_ICHAR *strtok(ZIL_ICHAR *s1, const ZIL_ICHAR *s2);
static ZIL_ICHAR *strerror(int errnum);

// OpenZinc (non-ANSI) routines
static int stricmp(const ZIL_ICHAR *a, const ZIL_ICHAR *b);
static int strnicmp(const ZIL_ICHAR *a, const ZIL_ICHAR *b, int n);
static ZIL_ICHAR *strlwr(ZIL_ICHAR *string);
static ZIL_ICHAR *strupr(ZIL_ICHAR *string);
static void strip(ZIL_ICHAR *string, ZIL_ICHAR c);
static ZIL_ICHAR *strdup(const ZIL_ICHAR *string);
static int WildStrcmp(ZIL_ICHAR *str, ZIL_ICHAR *pattern);
static void StripHotMark(ZIL_ICHAR *fillLine);

// Ansi routines
static int strlen(const ZIL_ICHAR *string);
static void itoa(ZIL_INT32 value, ZIL_ICHAR *string, int radix,
int pad = 0);
static void strepc(ZIL_ICHAR *string, int c, int repc);
static int sprintf(ZIL_ICHAR *buffer, const ZIL_ICHAR *format, ...);
static int sscanf(ZIL_ICHAR *buffer, const ZIL_ICHAR *format, ...);

// File support routines.
static int chdir(const ZIL_ICHAR *path);
static ZIL_ICHAR *getcwd(ZIL_ICHAR *buffer, unsigned length);
static ZIL_ICHAR *getenv(const ZIL_ICHAR *envname);
static int open(const ZIL_ICHAR *path, int access, unsigned mode = 0);
static int rename(const ZIL_ICHAR *oldPath, const ZIL_ICHAR *newPath);
static int stat(const ZIL_ICHAR *path, void *);
static ZIL_ICHAR *tmpnam(ZIL_ICHAR *path);
static int unlink(const ZIL_ICHAR *path);

```

```

// Character mapping routines
static char *MapText(const ZIL_ICHAR *mapped,
    char *unMapped = ZIL_NULLP(char), int allocate = TRUE);
static ZIL_ICHAR *UnMapText(const char *unMapped,
    ZIL_ICHAR *mapped = ZIL_NULLP(ZIL_ICHAR), int allocate = TRUE);
static ZIL_ICHAR *ISotoUNICODE(const char *isoString,
    ZIL_ICHAR *retValue = ZIL_NULLP(ZIL_ICHAR));

// File support routines.
static void ConvertFromFilename(ZIL_ICHAR *dst,
    const ZIL_FILE_CHAR *src);
static void ConvertToFilename(ZIL_FILE_CHAR *dst, const ZIL_ICHAR *src);

// Character mapping routines
static ZIL_ICHAR *ISotoICHAR(const char *isoString,
    ZIL_ICHAR *icharString = ZIL_NULLP(ZIL_ICHAR));
static int LoadICHARToHardware(const ZIL_ICHAR *mapName,
    const ZIL_ICHAR *extraName);
static ZIL_ICHAR UnMapChar(const char *hardware);
static char *MapChar(ZIL_ICHAR Unicode);

#if defined (ZIL_UNICODE)
static int mblen(const char *hardware);
static int wcstombs(char *s, const ZIL_ICHAR *pwcs, int n = -1);
static int mbstowcs(ZIL_ICHAR *pwcs, const char *s, int n = -1);
static ZIL_ICHAR *DecomposeCharacter(ZIL_ICHAR val);
static ZIL_ICHAR *DecomposeString(const ZIL_ICHAR *str);
#endif

// I18N member variables and functions,
public:
static void ParseLangEnv(ZIL_ICHAR *codeSet, ZIL_ICHAR *locName,
    ZIL_ICHAR *langName);
static const ZIL_LOCALE *defaultLocale;
static const ZIL_LOCALE canonicalLocale;
static ZIL_MAP_CHARS *defaultCharMap;

static ZIL_ICHAR _blankString[];
static ZIL_ICHAR _errorString[];

static void MachineName(void);
static ZIL_ICHAR machineName[32];
};

```

## General Members

This section describes those members used for general purposes.

- *minutesWestGMT* is how far west of Greenwich Mean Time the locale is, measured in minutes.

The use of the **Ctype**, **string**, and file support functions is the same as described for the C library with three exceptions. The first exception is that the **Ctype** functions are renamed to be consistent with OpenZinc coding standards (i.e., each "word" in the function name begins with a capital letter). This is done because most of these functions are actually implemented as macros in many compilers and need to be renamed to avoid symbol clashes.



The second difference is that the OpenZinc-overloaded functions use **ZIL\_ICHAR** instead of **char**. For a description of the **ZIL\_ICHAR** type, see "Appendix A—Support Definitions" of *Programmer's Reference Volume 2*.

The third difference is that the **printf** function has some enhanced formatting ability. If a '%n' is encountered in the formatting string, where n is an integer, the ordering of the fields will be altered so that the field will be the n-th field.

We will not discuss the use of these functions here. Refer to your C library reference for assistance with these functions.

We encourage you to use the OpenZinc implementation of these functions instead of the C library implementation even if you have no immediate plans to internationalize your applications. It will require little extra overhead and minimal extra effort, if any, but will provide a much easier path to internationalizing the application if you decide to do so in the future.

## ZIL\_INTERNATIONAL::CharMapInitialize

### Syntax

```
#include <ui_gen.hpp>

static void CharMapInitialize(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 new **ZIL\_MAP\_CHARS** object maintained by *defaultCharMap*.

## **ZIL\_ INTERNATIONAL::chartod**

### **Syntax**

```
#include <ui_gen.hpp>

static int chartod(const ZIL_ICHAR 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 returns the integer value of the character passed in. This function is used for determining the value of a number from the string representation of the number.

- *returnValue<sub>out</sub>* is the integer value of the character that was passed in. Thus, if the character was a digit, *returnValue* will be that digit's value. If the character that was passed in is not a digit, *returnValue* will be -1.
- *value<sub>in</sub>* is the character whose integer value is to be returned.

## **ZIL\_ INTERNATIONAL-ConvertFromFilename**

### **Syntax**

```
#include <ui_gen.hpp>

static void ConvertFromFilename(ZIL_ICHAR *dst, const ZIL_FILE_CHAR *src);
```

## Portability

This function is available on the following environments:

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

## Remarks

This function converts an environment-specific filename string to a Unicode filename string. A special function is necessary to convert filenames to ensure that any characters that may cause problems as part of a filename are not translated incorrectly (e.g., '/' or

The ZIL\_FILE\_CHAR type may be different in each environment, depending on the size of the character type in that environment.

- *dst<sub>out</sub>* is a pointer to a buffer where the converted filename will be placed. This buffer must be big enough to hold the converted filename.
- *src<sub>in</sub>* is a pointer to the filename string that is to be converted.

## ZIL INTERNATIONAL::ConvertToFilename

### Syntax

```
#include <ui_gen.hpp>
```

```
static void ConvertToFilename(ZIL_FILE_CHAR *dst, const ZIL_ICHAR *src);
```

## Portability

This function is available on the following environments:

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

## Remarks

This function converts a Unicode filename string to an environment-specific filename string. A special function is necessary to convert filenames to ensure that any characters that may cause problems as part of a filename are not translated incorrectly (e.g., '/' or

The `ZIL_FILE_CHAR` type may be different in each environment, depending on the size of the character type in that environment.

- `dstout` is a pointer to a buffer where the converted filename will be placed. This buffer must be big enough to hold the converted filename.
- `srcin` is a pointer to the string that is to be converted.

## ZIL\_INTERNATIONAL::DecomposeCharacter

### Syntax

```
#include <ui_gen.hpp>
```

```
static ZIL_ICHAR *DecomposeCharacter(ZIL_ICHAR val);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function splits a composed Unicode character into the individual characters it is made from. Some languages use characters as modifiers in certain contexts. One or more modifiers may be combined with other characters to form the desired character. It may be necessary to decompose the character in order to properly collate it. This functionality is only included in the library if the library is built with **ZIL\_DECOMPOSE** defined in **UI\_ENV.HPP**. If the application will not need this functionality, make sure **ZIL\_DECOMPOSE** is not defined and rebuild the library.

- `returnValueout` is the string formed from the individual characters obtained by decomposing `val`.
- `valin` is the Unicode character that is to be decomposed.

## ZIL\_INTERATIONAL: .DecomposeString

### Syntax

```
#include <ui_gen.hpp>

static ZIL_ICHAR *DecomposeString(const ZIL_ICHAR *str);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function decomposes the characters in a Unicode string. See the description of **DecomposeCharacter()**, above, for more details on composed characters. This functionality is only included in the library if the library is built with **ZIL\_DECOMPOSE** defined in **UI\_ENV.HPP**.

## ZIL\_INTERATIONAL::DefaultI18nInitialize

### Syntax

```
#include <ui_gen.hpp>

static void DefaultI18nInitialize(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 initializes the *defaultLocale* member. It is called from the constructor of the Event Manager as the program is starting up.

## ZIL INTERNATIONAL::I18nInitialize

### Syntax

```
#include <ui_gen.hpp>

static void I18nInitialize(const ZIL_ICHAR *localeName,
                          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 initializes the language and locale information being used by the application.

- *localeName<sub>in</sub>* is the two-letter ISO country code that identifies which locale is to be initialized.
- *languageName<sub>in</sub>* is the two-letter ISO language code that identifies which language is to be initialized.

## ZIL INTERNATIONAL::IsNonSpacing

### Syntax

```
#include <ui_gen.hpp>

static int IsNonSpacing(ZIL_ICHAR 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 determines if a character requires space when rendered. For example, some languages (e.g., Thai) contain characters that distinguish a related character in a string. The distinguishing character will appear above, below, before or after the character to be distinguished. Such a distinguishing character would be non-spacing since it is rendered in the region of the screen occupied by the distinguished character.

- *returnValue<sub>out</sub>* indicates whether the character is a spacing or non-spacing character. *returnValue* will be FALSE if the character is a spacing character, or TRUE if the character is non-spacing.
- *value<sub>in</sub>* is the Unicode character whose spacing requirements are to be determined.

## ZIL INTERNATIONAL:-.ISOtoICHAR

### Syntax

```
#include <ui_gen.hpp>

static ZIL_ICHAR *ISOtoICHAR(const char *isoString,
    ZIL_ICHAR *icharString = 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 function converts an 8-bit ISO8859-1 string to a Unicode string. The conversion is

performed by padding the high-order byte with zeros. This can be done because the first 256 entries in the Unicode character set are the ISO8859-1 characters. If no destination buffer is passed in, a buffer will be allocated by the function.

- *returnValue<sub>out</sub>* is a pointer to a buffer containing the generated Unicode string.
- *isoString<sub>in</sub>* is a pointer to the ISO8859-1 string that is to be converted.
- *unicodeString<sub>out</sub>* is a pointer to a buffer to which the generated Unicode string will be copied. If this pointer is used, the buffer must be big enough to hold the converted string. If no buffer is passed in the function will allocate a buffer. This buffer will need to be deleted by the programmer when he is done with the buffer.

## ZIL\_INTERNATIONAL::ISOtoUNICODE

### Syntax

```
#include <ui_gen.hpp>

static ZIL_ICHAR *ISOtoUNICODE(const char *isoString,
                                ZIL_ICHAR *returnValue = 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 function converts an 8-bit ISO8859-1 string to a Unicode string. The conversion is performed by padding the high-order byte with zeros. This can be done because the first 256 entries in the Unicode character set are the ISO8859-1 characters. If no destination buffer is passed in, a buffer will be allocated by the function.

- *returnValue<sub>out</sub>* is a pointer to a buffer containing the generated Unicode string.
- *isoString<sub>in</sub>* is a pointer to the ISO8859-1 string that is to be converted.



- *returnValue<sub>out</sub>* is a pointer to a buffer to which the generated Unicode string will be copied. If this pointer is used, the buffer must be big enough to hold the converted string. If no buffer is passed in the function will allocate a buffer. This buffer will need to be deleted by the programmer when he is done with the buffer.

## ZIL\_INTERATIONAL::LoadICHARtoHardware

### Syntax

```
#include <ui_gen.hpp>
```

```
static int LoadICHARtoHardware(const ZIL_ICHAR *mapName,
    const ZIL_ICHAR *extraName);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function loads map tables to provide mapping between the Unicode character set and the hardware character set. The hardware character set is the character set used by the operating system in the environment on which the program is running.

- *returnValue<sub>out</sub>* indicates whether the function was successful in loading the map tables. *returnValue* is 0 if the map tables were loaded successfully. It is non-zero if not successful.
- *mapName<sub>in</sub>* is the name of the map table to be loaded. This table is a standard table for mapping characters between the Unicode character set and the particular hardware character set.
- *extraName<sub>in</sub>* is the name of a map table that contains exceptions to the *mapName* map table. For instance, it is possible for the same character set on different operating systems to be slightly different (i.e., some characters may be in different locations in the character set). This map table resolves those problems by providing the proper mapping for known exceptions. This map table will be searched before the *mapName*

map table. If a mapping is found in this map table, the *mapName* table will not be searched.

## ZIL\_INTERNATIONAL::MapChar

### Syntax

```
#include <ui_gen.hpp>

static char *MapChar(ZIL_ICHAR Unicode);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function maps a character from the Unicode character set to the hardware character set. **MapChar** will allocate space for the resulting string which the programmer is responsible for deleting.

- *returnValue<sub>out</sub>* is a pointer to the hardware string.
- *Unicode<sub>in</sub>* is the Unicode character that is to be mapped to the hardware character set.

## ZIL\_INTERNATIONAL::MapText

### Syntax

```
#include <ui_gen.hpp>

static char *MapText(const ZIL_ICHAR *mapped, char *unmapped = ZIL_NULLP(char),
int allocate = 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 maps a string from the Unicode character set to the hardware character set. It does this by calling the **MapText()** function for the *defaultCharMap* member. **MapText** may allocate space for the resulting string which the programmer would be responsible for deleting.

- *returnValue<sub>out</sub>* is a pointer to the hardware string.
- *mapped<sub>in</sub>* is the Unicode string that is to be mapped to the hardware character set.
- *unmapped<sub>out</sub>* is a buffer in which the hardware string will be placed. If used, this buffer must be large enough to contain the string. If no buffer is passed in, the function can be directed to allocate a buffer or to use a temporary buffer.
- *allocate<sub>in</sub>* specifies if the function should allocate a buffer for the hardware string. If *allocate* is TRUE and no buffer was passed through the *unmapped* argument, a new buffer is allocated. Otherwise no buffer is allocated.

## ZIL\_INTERNATIONAL::mblen

### Syntax

```
#include <ui_gen.hpp>
```

```
static int mblen(const char *hardware);
```

## Portability

This function is available on the following environments:

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

## Remarks

This function calculates how many bytes long a character is. The characters in some character sets may be from one to four bytes long. This function is used to determine the length of the first character in a string of multi-byte characters. This function calls the **mblen()** function of the *defaultCharMap* member.

- *returnValue<sub>out</sub>* is the number of bytes in the first character of the string passed in.
- *hardware<sub>in</sub>* is the multi-byte character string of which the first character's size is required.

## ZIL INTERNATIONAL::mbstowcs

### Syntax

```
#include <ui_gen.hpp>
```

```
static int mbstowcs(ZIL_ICHAR *pwcs, const char *s, int n = -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 converts a multi-byte character string to a wide-character Unicode string. The source string is made up of characters from any hardware character set. This function calls the **mbstowcs()** function of the *defaultCharMap* member.

- *returnValue* is a count of how many characters are in the converted string.
- *pwcs<sub>out</sub>* is a pointer to a buffer in which the converted wide-character Unicode string will be placed. This buffer must be large enough to contain the string.
- *s<sub>in</sub>* is the source string to be converted. This string is made up of characters from the local hardware character set.

- $n_{in}$  is a count of how many characters are to be converted. If  $n$  is less than 0, the `strlen` of 5 is used, `strlen` will provide the maximum number of characters that the input string may contain since `strlen` will not necessarily return the number of actual characters in the string, but rather the number of 8-bit values in the string. Some characters may be more than 8-bits wide.

## **ZIL INTERNATIONAL::OSI18nInitialize**

### **Syntax**

```
#include <ui_gen.hpp>
```

```
static void OSI18nInitialize(ZIL_ICHAR *langName, int forceInitialization = 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 loads language and locale data from the operating system.

- *langName<sub>out</sub>* is the two-letter ISO language name in use by the operating system. *langName* is a buffer that should be allocated by the programmer.
- *forceInitialization<sub>in</sub>* specifies what the function should do if the locale data has already been initialized. If *forceInitialization* is TRUE, the locale data will be loaded from the operating system even if it has already been initialized. If *forceInitialization* is FALSE, the locale data will not be loaded from the operating system if it has already been initialized. This function is called from the constructor of the Event Manager as the program is starting up.

## ZIL\_INTERATIONAL::StripHotMark

### Syntax

```
#include <ui_gen.hpp>
```

```
static void StripHotMark(ZIL_ICHAR *fillLine);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function strips the hotkey marker (i.e., '&') from a string.

- *fillLine<sub>in/out</sub>* is the string that is to have its hotkey markers removed. The new string is placed back into the existing buffer after having the hotkey markers removed.

## ZIL\_INTERATIONAL::strup

### Syntax

```
#include <ui_gen.hpp>
```

```
static voidstrup(ZIL_ICHAR "string, ZIL_ICHAR c);
```

### Portability

This function is available on the following environments:

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

## Remarks

This function strips all occurrences of *c* from *string*.

- *string<sub>in/out</sub>* is the string that is to be stripped.
- *c<sub>in</sub>* is the character that is to be removed from *string*.

## ZIL\_INTERNATIONAL::TimeStamp

### Syntax

```
#include <ui_gen.hpp>

void TimeStamp(ZIL_UINT32 *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 returns the current time specified in milliseconds since January 1, 1970.

- *value<sub>out</sub>* is the number of milliseconds that have passed since January 1, 1970.

## ZIL\_INTERNATIONAL::UnMapChar

### Syntax

```
#include <ui_gen.hpp>

static ZIL_ICHAR UnMapChar(const char *hardware);
```

## Portability

This function is available on the following environments:

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

## Remarks

This function unmaps a character from the hardware character set and returns the equivalent character from the Unicode character set. It does this by calling the `UnMapChar( )` function for the `defaultCharMap` member.

- `returnValueout` is the unmapped Unicode character.
- `hardwarein` is a pointer to a buffer containing the mapped hardware character.

## ZIL INTERNATIONAL::UnMapText

### Syntax

```
#include <ui_gen.hpp>

static ZIL_ICHAR *UnMapText(const char *unmapped,
    ZIL_ICHAR *mapped=ZIL_NULLP(ZIL_ICHAR), int allocate=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 unmaps a string from a hardware character set and restores it to a Unicode character string.

- `returnValueout` is a pointer to the unmapped Unicode text.



- *unmapped<sub>in</sub>* is a pointer to a buffer containing the mapped hardware text.
- *mapped<sub>out</sub>* is a pointer to a buffer in which the unmapped Unicode text will be placed.
- *allocate<sub>in</sub>* indicates if the function should allocate a buffer for the unmapped Unicode text. If *allocate* is TRUE and no buffer is passed through the *mapped* argument, the function will allocate a buffer. This buffer must be deleted by the programmer when he is done with it.

## **ZIL\_INTERATIONAL::wcstombs**

### **Syntax**

```
#include <ui_gen.hpp>
```

```
static int wcstombs(char *s, const ZIL_ICHAR *pwcs, int n = -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 converts a wide-character Unicode string to a multi-byte character string. The destination string will be made up of characters from the hardware character set. This function calls the **wcstombs()** function of the *defaultCharMap* member.

- *returnValue* is a count of how many characters are in the converted string.
- *s<sub>out</sub>* is a pointer to a buffer in which the multi-byte character string will be placed after it has been converted from the Unicode string. This string will be made up of characters from the local hardware character set. The buffer must be large enough to hold the converted string.
- *pwcs<sub>in</sub>* is a pointer to the wide-character Unicode string.

- $n_{in}$  is a count of how many characters are to be converted. If  $n$  is less than 0, the **strlen** of  $s$  is used.

## **ZIL\_INTERNATIONAL::WildStrcmp**

### **Syntax**

```
#include <ui_gen.hpp>
```

```
static int WildStrcmp(ZIL_ICHAR *str, ZIL_ICHAR *pattern);
```

### **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 a **strcmp** that can accommodate wild card characters (i.e., '\*' and '?').

- $returnValue_{out}$  indicates the result of the compare. If the two strings matched,  $returnValue$  will be 0. Otherwise it will be a non-zero value.
- $str_{in}$  is a pointer to the string that is to be compared against.
- $pattern_{in}$  is a pointer to the string that contains the pattern to be compared for.

## **Internationalization Members**

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

- *defaultLocale* is the default ZIL\_LOCALE object that is used to format data for a particular locale.
- *canonicalLocale* is a ZIL\_LOCALE class that is used when converting objects from one locale to another, particularly when storing the object in a file. A consistent

format must be used so that data can be properly interpreted regardless of which locale the data was formatted for when stored or which locale it needs to be formatted for when loaded, *canonicalLocale* provides the consistent formatting information.

- *defaultCharMap* is the `ZIL_MAP_CHARS` object that is used to map characters among character sets.
- *\_blankString* is an empty string.
- *\_errorString* is the default error message.
- *machineName* is a string that identifies the type of hardware on which the program is running.

## ZIL INTERNATIONAL::*MachineName*

### Syntax

```
#include <ui_gen.hpp>

static void MachineName(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 identifies the hardware platform being used and sets *machineName* accordingly. This function is available in DOS mode only.

## ZIL\_INTERNATIONAL::ParseLangEnv

### Syntax

```
#include <ui_gen.hpp>

static void ParseLangEnv(ZIL_ICHAR *codeSet, ZIL_ICHAR *_locName,
                        ZIL_ICHAR *_langName)\
```

### Portability

This function is available on the following environments:

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

### Remarks

This function queries the operating system to learn which language, locale and character set should be used. It determines which to set up by inspecting the *ZINC\_LANG* environment variable, if it has been defined. The value of *ZINC\_LANG* should have the format *language[\_locale][.codeSet]*, where *language* is the two-letter ISO language code to be used, *\_locale* is the two-letter ISO country code to be used and *.codeSet* is the character set to be used for the hardware character set.

- *codeSet<sub>out</sub>* is a string identifying which character set should be used. The buffer pointed to by *codeSet* must be allocated by the programmer.
- *\_locName<sub>out</sub>* is the two-letter ISO country code specified in the *ZINC\_LANG* environment variable. The buffer pointed to by *\_locName* must be allocated by the programmer.
- *\_langName<sub>out</sub>* is the two-letter ISO language code specified in the *ZINC\_LANG* environment variable. The buffer pointed to by *\_langName* must be allocated by the programmer.



## CHAPTER 59 - ZIL\_LANGUAGE

The ZIL\_LANGUAGE class object is used to maintain language translations for an object. Any object that needs string translations has a pointer to the ZIL\_LANGUAGE object containing that object's translations. The object can get a pointer to the appropriate ZIL\_LANGUAGE object through the ZIL\_LANGUAGE\_MANAGER, which maintains a list of all ZIL\_LANGUAGE objects. The strings are each kept in a ZIL\_LANGUAGE\_ELEMENT. Because each instance of an object can have its own ZIL\_LANGUAGE object, any combination of languages can be used simultaneously.

The ZIL\_LANGUAGE class is declared in UI\_GEN.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS ZIL_LANGUAGE : public ZIL_I18N
{
public:
    ZIL_LANGUAGE(void);
#ifdef ZIL_LOAD
    ZIL_LANGUAGE(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));
    virtual void Load(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));
    void Load(ZIL_STORAGE_READ_ONLY *storage,
              ZIL_STORAGE_OBJECT_READ_ONLY *object);
    virtual void ClassLoadData(ZIL_STORAGE_OBJECT_READ_ONLY *object);
#endif
#ifdef 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));
    void Store(ZIL_STORAGE *storage, ZIL_STORAGE_OBJECT *object);
    virtual void ClassStoreData(ZIL_STORAGE_OBJECT *object);
#endif
    int noOfElements;
    ZIL_LANGUAGE_ELEMENT *data;
    ZIL_ICHAR *GetMessage(ZIL_NUMBERID numberID,
                        int useDefault = FALSE) const;
protected:
    virtual void AssignData(const ZIL_I18N *data);
    virtual void DeleteData(void);
};
```

### General Members

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

- *noOfElements* indicates how many strings are maintained by the ZIL\_LANGUAGE object.
- *data* is the list of ZIL\_LANGUAGE\_ELEMENT objects that contain the translated strings.

## **ZIL\_LANGUAGE::ZIL\_LANGUAGE**

### **Syntax**

```
#include <ui_gen.hpp>

ZIL_LANGUAGE( void);
```

### **Remarks**

This constructor creates a new ZIL\_LANGUAGE object.

## **ZIL\_LANGUAGE::AssignData**

### **Syntax**

```
#include <ui_gen.hpp>

virtual void AssignData(const ZIL_I18N *data);
```

### **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 the data maintained by *data* to the ZIL\_LANGUAGE object. The *noOfElements* value and the *data* pointer are copied. This function does not create a new copy of the data, but simply assigns the pointer.

- *data<sub>in</sub>* is a pointer to the ZIL\_I18N object containing the data that is to be assigned.

## ZIL\_LANGUAGE::DeleteData

### Syntax

```
#include <ui_gen.hpp>

virtual void DeleteData(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 deletes the data maintained by this object if *allocated* is TRUE. The text in each ZIL\_LANGUAGE\_ELEMENT as well as the list of ZIL\_LANGUAGE\_ELEMENT objects is deleted.

## ZIL\_LANGUAGE::GetMessage

### Syntax

```
#include <ui_gen.hpp>

ZIL_ICHAR *GetMessage(ZIL_NUMBERID numberID,
    int useDefault = FALSE) const;
```

### 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 string maintained by the ZIL\_LANGUAGE\_ELEMENT object identified by *numberID*.

- *returnValue<sub>out</sub>* is the translated string.
- *nurnberID<sub>in</sub>* is a value identifying the ZIL\_LANGUAGE\_ELEMENT.
- *useDefault<sub>in</sub>* indicates if the default text should be used if no match is found on *numberID*. If *useDefault* is TRUE, the text of the first ZIL\_LANGUAGE\_ELEMENT is returned if no ZIL\_LANGUAGE\_ELEMENT objects matched *numberID*. If no match was found and *useDefaults* is FALSE, NULL is returned.

## Storage Members

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

## ZIL\_LANGUAGE::ZIL\_LANGUAGE

### Syntax

```
#include <ui_gen.hpp>

ZIL_LANGUAGE(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));
```

### 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 is used to load ZIL\_LANGUAGE data from a persistent object data file. It is typically not used by the programmer.

- *name<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* is a pointer to the ZIL\_STORAGE\_READ\_ONLY object that contains the data. For more information on persistent object files, see "Chapter 70—ZIL\_STORAGE\_READ\_ONLY."
- *object<sub>in</sub>* is a pointer to the ZIL\_STORAGE\_OBJECT\_READ\_ONLY where the data will be loaded. For more information on loading information from persistent object files, see "Chapter 69—ZIL\_STORAGE\_OBJECT\_READ\_ONLY."

## ZIL\_LANGUAGE::ClassLoadData

### Syntax

```
#include <ui_gen.hpp>
```

```
virtual void ClassLoadData(ZIL_STORAGE_OBJECT_READ_ONLY *object);
```

### 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 ZIL\_LANGUAGE data from a persistent object data file. The data is loaded from the current directory. This function is typically not used by the programmer.

- *object<sub>in</sub>* is a pointer to the ZIL\_STORAGE\_OBJECT\_READ\_ONLY object that contains the data. For more information on persistent objects, see "Chapter 69—ZIL\_STORAGE\_OBJECT\_READ\_ONLY."

## ZIL\_LANGUAGE::ClassStoreData

### Syntax

```
#include <ui_gen.hpp>

virtual void ClassStoreData(ZIL_STORAGE_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 advanced function is used to store ZIL\_LANGUAGE data in a persistent object data file. The data is stored in the current directory. This function is typically not used by the programmer.

- *object<sub>in</sub>* is a pointer to the ZIL\_STORAGE\_OBJECT where the data will be stored. For more information on persistent objects, see "Chapter 68—ZIL\_STORAGE\_OBJECT."

## ZIL\_LANGUAGE::Load

### Syntax

```
#include <ui_gen.hpp>

virtual void Load(const ZIL_ICHAR *name,
  ZIL_STORAGE_READ_ONLY *fie =
  ZIL_NULLP(ZIL_STORAGE_READ_ONLY),
  ZIL_STORAGE_OBJECT_READ_ONLY *object =
  ZIL_NULLP(ZIL_STORAGE_OBJECT_READ_ONLY));
or
void Load(ZIL_STORAGE_READ_ONLY *storage,
  ZIL_STORAGE_OBJECT_READ_ONLY *object);
```

## 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 ZIL\_LANGUAGE data from a persistent object data file. This function is typically not used by the programmer.

- *name<sub>in</sub>* is the name of the object to be loaded.
- *file<sub>in</sub>* and *storage<sub>in</sub>* are pointers to the ZIL\_STORAGE\_READ\_ONLY object that contains the data. For more information on persistent object files, see "Chapter 70—ZIL\_STORAGE\_READ\_ONLY."
- *object<sub>in</sub>* is a pointer to the ZIL\_STORAGE\_OBJECT\_READ\_ONLY where the data will be loaded. For more information on loading information from persistent object files, see "Chapter 69—ZIL\_STORAGE\_OBJECT\_READ\_ONLY."

## ZIL\_LANGUAGE::Store

### Syntax

```
#include <ui_gen.hpp>

virtual void Store(const ZIL_ICHAR *name,
    ZIL_STORAGE *file = ZIL_NULLP(ZIL_STORAGE),
    ZIL_STORAGE_OBJECT *object = ZIL_NULLP(ZIL_STORAGE_OBJECT));
    or
void Store(ZIL_STORAGE *storage, ZIL_STORAGE_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 advanced function is used to store ZIL\_LANGUAGE data in a persistent object data file. This function is typically not used by the programmer.

- *name<sub>in</sub>* is the name of the object to be stored.
- *file<sub>in</sub>* and *storage<sub>in</sub>* are pointers to the ZIL\_STORAGE where the data will be stored. For more information on persistent object files, see "Chapter 66—ZIL\_STORAGE."
- *object<sub>in</sub>* is a pointer to the ZIL\_STORAGE\_OBJECT where the data will be stored. For more information on loading persistent objects, see "Chapter 68—ZIL\_STORAGE\_OBJECT."

## CHAPTER 60 - ZIL\_LANGUAGE\_ELEMENT

The ZIL\_LANGUAGE\_ELEMENT structure is used by the ZIL\_LANGUAGE class to provide a translated text string.

The ZIL\_LANGUAGE\_ELEMENT class is declared in UI\_GEN.HPP. Its public and protected members are:

```
Struct ZIL_EXPORT_CLASS ZIL_LANGUAGE_ELEMENT
{
    ZIL_ICHAR *text;
    ZIL_NUMBERID numberID;
    ZIL_ICHAR stringID[ZIL_STRINGID_LEN];

    void SwapData(ZIL_LANGUAGE_ELEMENT &language);
};
```

### General Members

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

- *text* is the text maintained by the ZIL\_LANGUAGE\_ELEMENT.
- *numberID* is a numeric value used to identify the ZIL\_LANGUAGE\_ELEMENT.
- *stringID* is a string value used to identify the ZIL\_LANGUAGE\_ELEMENT.

### ZIL\_LANGUAGE\_ELEMENT::SwapData

#### Syntax

```
#include <ui_gen.hpp>
```

```
void SwapData(ZIL_LANGUAGE_ELEMENT &language);
```

#### Portability

This function is available on the following environments:

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

## Remarks

This function swaps the contents of the ZIL\_LANGUAGE\_ELEMENT with the contents of *language*.

- *language<sub>in</sub>* is the ZIL\_LANGUAGE\_ELEMENT whose contents are to be swapped with the ZIL\_LANGUAGE\_ELEMENT.

## CHAPTER 61 - ZIL\_LANGUAGE\_MANAGER

The ZIL\_LANGUAGE\_MANAGER class object is used to maintain a list of ZIL\_LANGUAGE objects. Each ZIL\_LANGUAGE class contains translations for a particular language for a single object.

The ZIL\_LANGUAGE\_MANAGER class is declared in UI\_GEN.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS ZIL_LANGUAGE_MANAGER : public ZIL_I18N_MANAGER
{
public:
    ZIL_LANGUAGE_MANAGER(void);
    virtual ZIL_I18N *CreateData(void);

    static void FreeLanguage(const ZIL_LANGUAGE *language);
    static void LoadDefaultLanguage(const ZIL_ICHAR *languageName);
    static const ZIL_LANGUAGE *UseLanguage(const ZIL_LANGUAGE *language);
    static const ZIL_LANGUAGE *UseLanguage(const ZIL_ICHAR *className,
        const ZIL_ICHAR *languageName = ZIL_NULLP(ZIL_ICHAR));

    static void SetLanguage(const ZIL_ICHAR *className,
        ZIL_PRIVATE_LANGUAGE_ELEMENT *defaultMessages);
};
```

### General Members

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

### ZIL\_LANGUAGE\_MANAGER::ZIL\_LANGUAGE\_MANAGER

#### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_LANGUAGE_MANAGER(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 ZIL\_LANGUAGE\_MANAGER object.

## ZIL LANGUAGE MANAGER::Create Data

### Syntax

```
#include <ui_gen.hpp>

virtual ZIL_I18N *CreateData(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 creates a new ZIL\_LANGUAGE object. Because it is a pure virtual function at the base ZIL\_I18N\_MANAGER class level, the generic code in the ZIL\_I18N\_MANAGER class can use it to create the proper data object.

- *returnValue<sub>out</sub>* is a pointer to the new ZIL\_LANGUAGE object that was created.

## ZIL LANGUAGE MANAGER::FreeLanguage

### Syntax

```
#include <ui_gen.hpp>

static void FreeLanguage(const ZIL_LANGUAGE *language);
```

## Portability

This function is available on the following environments:

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

## Remarks

This function releases the ZIL\_LANGUAGE object from use by decrementing the ZIL\_LANGUAGE object's *useCount* member. Whenever a library object requests the use of a ZIL\_ILANGUAGE object, it does so by calling the **UseLanguage()** function, which marks the ZIL\_LANGUAGE object as used by incrementing the its *useCount* member. When the library object is done using the ZIL\_LANGUAGE object, it must release it by calling this function. If the releasing object was the last object using the ZIL\_LANGUAGE object, this function will deallocate the data being maintained by the ZIL\_LANGUAGE object unless it contains the default data.

- *language<sub>in</sub>* is a pointer to the ZIL\_LANGUAGE object that is being released.

## ZIL\_LANGUAGE\_MANAGER::LoadDefaultLanguage

### Syntax

```
#include <ui_gen.hpp>
```

```
static void LoadDefaultLanguage(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 default language for the application. Any objects that are using the default language information at the time this function is called will start using the new

default data. If necessary, this function loads the default language data from the **I18N.DAT** file.

- *languageName<sub>in</sub>* is the two-letter ISO name identifying the language which is to be the default for the application.

## ZIL\_LANGUAGE\_MANAGER::SetLanguage

### Syntax

```
#include <ui_gen.hpp>

static void SetLanguage(const ZIL_ICHAR *className,
    ZIL_PRIVATE_LANGUAGE_ELEMENT *defaultMessages);
```

### 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 default translations for a particular object in the list of translations maintained by the ZIL\_LANGUAGE\_MANAGER class. A translation must be placed in the list using this function before the translations can be accessed using the **UseLanguage()** function. The translations are assumed to be in the language identified by the *isoLanguageName* global variable. This variable and the default translations are defined in the **LANG\_DEF.CPP** file. If a different default language is desired, simply copy a **LANG\_<ISO>.CPP** file from the ZINC\SOURCE\INTL directory to the ZINCV SOURCE directory and rename it to **LANG\_DEF.CPP**. Then rebuild the library.

- *className<sub>in</sub>* is the class name of the object for which the language data is being set. This typically corresponds to the *\_className* member variable of the object.
- *defaultMessages<sub>in</sub>* is the ZIL\_PRIVATE\_LANGUAGE\_ELEMENT class that contains the translations for the object. The ZIL\_PRIVATE\_LANGUAGE\_ELEMENT class is simply a ZIL\_LANGUAGE\_ELEMENT class except that in Unicode mode it does not have the **SwapData( )** member function.

## ZIL\_LANGUAGE\_MANAGER::UseLanguage

### Syntax

```
#include <ui_gen.hpp>

static const ZIL_LANGUAGE *UseLanguage(const ZIL_LANGUAGE *language);
    or
static const ZIL_LANGUAGE *UseLanguage(const ZIL_ICHAR *className,
    const ZIL_ICHAR *languageName = 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

These overloaded functions mark a `ZIL_LANGUAGE` object as used by incrementing its `useCount` member. Whenever a library object requests the use of a `ZIL_LANGUAGE` object, it marks the object as used by calling this function, which increments the object's `useCount` member. When the library object is done using the `ZIL_LANGUAGE` object, it must release it by calling the **FreeLanguage()** function.

The first overloaded function takes a pointer to the `ZIL_LANGUAGE` object being marked as used.

- `returnValueout` is a pointer to the `ZIL_LANGUAGE` object.
- `languagein` is a pointer to the `ZIL_LANGUAGE` object that is to be marked as used.

The second overloaded function takes the class name of the object and the internationalization name. If load capability is enabled (i.e., **ZIL\_LOAD** was defined when the library was compiled) this function will load the data from the **I18N.DAT** file if necessary. Otherwise, the data must have been compiled and linked into the application.

- `returnValueout` is a pointer to the `ZIL_ILANGUAGE` object.

- *className<sub>in</sub>* is the class name of the object for which the internationalization data is being requested. This typically corresponds to the *\_className* member variable of the object.
- *languageName<sub>in</sub>* is the two-letter ISO name identifying the language to be used.

## CHAPTER 62 - ZIL LOCALE

The ZIL\_LOCALE class object is used to maintain locale information for a particular country for all objects. Any object that needs locale information has a pointer to the ZIL\_LOCALE object containing the data for the desired country. The object can get a pointer to the appropriate ZIL\_LOCALE object through the ZIL\_LOCALE\_MANAGER, which maintains a list of all ZIL\_LOCALE objects. Because each instance of an object can have its own ZIL\_LOCALE object, each field can be formatted for a different locale, if desired.

The ZIL\_LOCALE class is declared in **UI\_GEN.HPP**. Its public and protected members are:

```
class ZIL_EXPORT_CLASS ZIL_LOCALE : public ZIL_I18N
{
public:
    ZIL_LOCALE(void);
    #if defined(ZIL_LOAD)
    virtual void ClassLoadData(ZIL_STORAGE_OBJECT_READ_ONLY *object);
    #endif
    #if defined(ZIL_STORE)
    virtual void ClassStoreData(ZIL_STORAGE_OBJECT *object);
    #endif
    ZIL_LOCALE_ELEMENT data;
    #if defined(ZIL_MSDOS) || defined(ZIL_MSWINDOWS) || defined(ZIL_OS2) ||
    defined(ZIL_CURSES) || defined(ZIL_MACINTOSH)
    static int oemCountryCode;
    #endif
protected:
    virtual void AssignData(const ZIL_I18N *data);
    virtual void DeleteData(void);
};
```

### General Members

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

- *data* is the ZIL\_LOCALE\_ELEMENT object that contains the locale data for this locale.
- *oemCountryCode* is the country code for the environment on which the application is running. This value is used to set the locale data.

## ZIL\_LOCALE::ZIL\_LOCALE

### Syntax

```
#include <ui_gen.hpp>

ZIL_LOCALE(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 ZIL\_LOCALE class object.

## ZIL\_LOCALE::AssignData

### Syntax

```
#include <ui_gen.hpp>

virtual void AssignData(const ZIL_I18N *data);
```

### 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 the data maintained by *data* to the ZIL\_LOCALE object. The *data* is copied.

- *data<sub>in</sub>* is a pointer to the ZIL\_LOCALE object containing the data that is to be assigned.

## **ZIL\_LOCALE::DeleteData**

### **Syntax**

```
#include <ui_gen.hpp>

virtual void DeleteData(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 deletes the data maintained by this object.

## **Storage Members**

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

## **ZIL\_LOCALE::ClassLoadData**

### **Syntax**

```
#include <ui_gen.hpp>

virtual void ClassLoadData(ZIL_STORAGE_OBJECT_READ_ONLY *object);
```



## 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 ZIL\_LOCALE data from a persistent object data file. The data is loaded from the current directory. This function is typically not used by the programmer.

- *object<sub>in</sub>* is a pointer to the ZIL\_STORAGE\_OBJECT\_READ\_ONLY object that contains the data. For more information on persistent objects, see "Chapter 69—ZIL\_STORAGE\_OBJECT\_READ\_ONLY."

## ZIL\_LOCALE::ClassStoreData

### Syntax

```
#include <ui_gen.hpp>
```

```
virtual void ClassStoreData(ZIL_STORAGE_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 advanced function is used to store ZIL\_LOCALE data in a persistent object data file. The data is stored in the current directory. This function is typically not used by the programmer.

- *object<sub>in</sub>* is a pointer to the ZIL\_STORAGE\_OBJECT where the data will be stored. For more information on persistent objects, see "Chapter 68—ZIL\_STORAGE\_OBJECT."



## CHAPTER 63 - ZIL\_LOCALE\_ELEMENT

The ZIL\_LOCALE\_ELEMENT structure is used by the ZIL\_LOCALE class to provide locale information for objects.

The ZIL\_LOCALE\_ELEMENT structure is declared in UI\_GEN.HPP. Its public and protected members are:

```
struct ZIL_EXPORT_CLASS ZIL_LOCALE_ELEMENT
{
    ZIL_ICHAR decimalSeparator[4];
    ZIL_ICHAR monDecimalSeparator[4];
    ZIL_ICHAR thousandsSeparator[4];
    ZIL_ICHAR monThousandsSeparator[4];
    ZIL_ICHAR currencySymbol[8];
    char grouping[10];
    char monGrouping[10];
    ZIL_rCHAR intCurrencySymbol[5];
    int posCurrencyPrecedes;
    int negCurrencyPrecedes;
    int fractionDigits;
    int intFractionDigits;
    ZIL_ICHAR positiveSign[4];
    int posSignPrecedes;
    int posSpaceSeparation;
    ZIL_ICHAR negativeSign[4];
    int negSignPrecedes;
    int negSpaceSeparation;

    ZIL_ICHAR *bnumLeftParen;
    ZIL_ICHAR *bnumRightParen;

    ZIL_ICHAR *timeStringFormat;
    ZIL_ICHAR *dateStringFormat;
    ZIL_ICHAR *dateTimeStringFormat;
    ZIL_ICHAR *time12StringFormat;
    ZIL_ICHAR *defDigits;
    ZIL_ICHAR *altDigits;

    ZIL_ICHAR timeSeparator[4];
    ZIL_ICHAR dateSeparator[4];
    int defaultDateFlags;
    int defaultTimeFlags;
};
```

### General Members

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

NOTE: The following examples use the United States format unless otherwise specified.

- *decimalSeparator* is the system's decimal separator (e.g., 100.00).

- *monDecimalSeparator* is the system's currency decimal separator (e.g., \$100.00).
- *thousandsSeparator* is the system's thousands separator (e.g., 100,000).
- *monThousandsSeparator* is the system's currency thousands separator (e.g., \$100,000.00).
- *currencySymbol* is the system's currency symbol (e.g., '\$').
- *grouping* is a string that indicates the format to be used for grouping digits in a number. For more specific information, see the ANSI Standard Specification for the C Programming Language.
- *monGrouping* is a string that indicates the format to be used for grouping digits in a monetary number. For more specific information, see the ANSI Standard Specification for the C Programming Language.
- *intCurrencySymbol* is the system's international currency symbol (e.g., 'USD').
- *posCurrencyPrecedes* is TRUE if the currency symbol precedes the currency amount for positive currency values (e.g., '\$100,000.00'). Otherwise, *posCurrencyPrecedes* is FALSE and the currency symbol will be displayed following the currency amount.
- *negCurrencyPrecedes* is TRUE if the currency symbol precedes the currency amount for negative currency values (e.g., '-\$100,000.00'). Otherwise, *negCurrencyPrecedes* is FALSE and the currency symbol will be displayed following the currency amount.
- *fractionDigits* specifies the number of fractional digits to display after the decimal point on currency values (e.g., \$100.00).
- *intFractionDigits* specifies the number of fractional digits to display after the decimal point on currency values in international format (e.g., USD100.00).
- *positiveSign* specifies the symbol for positive values (e.g., +100 or 100).
- *posSignPrecedes* is TRUE if the symbol for positive amounts is displayed before the currency symbol. Otherwise, *posSignPrecedes* is FALSE and the symbol will be displayed following the amount.
- *posSpaceSeparation* is TRUE if there is a space separator between the currency symbol and the currency amount when the currency is positive. Otherwise, *posSpaceSeparation* is FALSE.

- *negativeSign* specifies the symbol for negative values (e.g., -100).
- *negSignPrecedes* is TRUE if the symbol for negative amounts is displayed before the currency symbol. Otherwise, *negSignPrecedes* is FALSE and the symbol will be displayed following the amount.
- *negSpaceSeparation* is TRUE if there is a space separator between the currency symbol and the currency amount when the currency is negative. Otherwise, *negSpaceSeparation* is FALSE.
- *bnumLeftParen* is the symbol used to the left of a negative number if the number format calls for encompassing negative symbols.
- *bnumRightParen* is the symbol used to the right of a negative number if the number format calls for encompassing negative symbols.
- *timeStringFormat* is the system's current time format (e.g., **12:00a**).
- *dateStringFormat* is the system's current date format (e.g., **12/25/91**).
- *dateTimeStringFormat* is the system's current format for presenting a date and time together.
- *time12StringFormat* is the system's current format for presenting a time.
- *defDigits* is a string containing the default digits to use when a '%d' is encountered in a **printf** format string.
- *altDigits* is a string containing the alternate digits to use when a '%ad' is encountered in a **printf** format string.
- *timeSeparator* is the system's time separator (e.g., 12:00).
- *dateSeparator* is the system's date separator (e.g., 12/25/91).
- *defaultDateFlags* is the default date flags as obtained from the operating system.
- *defaultTimeFlags* is the default time flags as obtained from the operating system.



## CHAPTER 64 - ZIL\_LOCALE\_MANAGER

The ZIL\_LOCALE\_MANAGER class object is used to maintain a list of ZIL\_LOCALE objects. Each ZIL\_LOCALE class contains all formatting information for a particular locale.

The ZIL\_LOCALE\_MANAGER class is declared in UI\_GEN.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS ZIL_LOCALE_MANAGER : public ZIL_I18N_MANAGER
{
public:
    ZIL_LOCALE_MANAGER(void);
    virtual ZIL_I18N *CreateData(void);

    static void FreeLocale(const ZIL_LOCALE *locale);
    static void LoadDefaultLocale(const ZIL_ICHAR *localeName);
    static const ZIL_LOCALE *UseLocale(const ZIL_LOCALE *locale);
    static const ZIL_LOCALE *UseLocale(const ZIL_ICHAR *localeName =
        ZIL_NULLP(ZIL_ICHAR));

    static void SetLocale(const ZIL_ICHAR *localeName,
        ZIL_LOCALE_ELEMENT *defaultLocale);
};
```

### General Members

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

### ZIL\_LOCALE\_MANAGER::ZIL\_LOCALE\_MANAGER

#### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_LOCALE_MANAGER(void);
```

#### Portability

This function is available on the following environments:

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



## Remarks

This constructor creates a new `ZIL_LANGUAGE_MANAGER` object.

## ZIL\_LOCALE\_MANAGER::CreateData

### Syntax

```
#include <ui_gen.hpp>

virtual ZIL_I18N *CreateData(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 creates a new `ZIL_LOCALE` object. Because it is a pure virtual function at the base `ZIL_I18N_MANAGER` class level, the generic code in the `ZIL_I18N_MANAGER` class can use it to create the proper data object.

- *returnValue*<sub>out</sub> is a pointer to the new `ZIL_LOCALE` object that was created.

## ZIL\_LOCALE\_MANAGER::FreeLocale

### Syntax

```
#include <ui_gen.hpp>

static void FreeLocale(const ZIL_LOCALE *locale);
```

## Portability

This function is available on the following environments:

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

## Remarks

This function releases the ZIL\_LOCALE object from use by decrementing the ZIL\_LOCALE object's *useCount* member. Whenever a library object requests the use of a ZIL\_LOCALE object, it does so by calling the **UseLocale()** function, which marks the ZIL\_LOCALE object as used by incrementing its *useCount* member. When the library object is done using the ZIL\_LOCALE object, it must release it by calling this function. If the releasing object was the last object using the ZIL\_LOCALE object, this function will deallocate the data being maintained by the ZIL\_LOCALE object unless it contains the default data.

- *locale<sub>in</sub>* is a pointer to the ZIL\_LOCALE object that is being released.

## **ZIL\_LOCALE\_MANAGER::LoadDefaultLocale**

### Syntax

```
#include <ui_gen.hpp>

static void LoadDefaultLocale(const ZIL_ICHAR *localeName)
```

## 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 default locale for the application. Any objects that are using the default locale information at the time this function is called will start using the new

default data. If necessary, this function loads the default locale data from the **I18N.DAT** file.

- *localeName<sub>in</sub>* is the two-letter ISO name identifying the locale which is to be the default for the application.

## ZIL LOCALE MANAGER::SetLocale

### Syntax

```
#include <ui_gen.hpp>

static void SetLocale(const ZIL_ICHAR *localeName,
                     ZIL_LOCALE_ELEMENT *defaultLocale)\
```

### 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 default locale data for a particular object in the list of locale information maintained by the ZIL\_LOCALE\_MANAGER class. Locale information must be placed in the list using this function before it can be accessed using the **UseLocale()** function. The default locale information is defined in the **LOC\_DEF.CPP** file. If a different default locale is desired, simply copy a **LOC\_<ISO>.CPP** file from the ZINC\SOURCE\INTL directory to the ZINC\SOURCE directory and rename it to **LOC\_DEF.CPP**. Then rebuild the library.

- *className<sub>in</sub>* is the class name of the object for which the locale data is being set. This typically corresponds to the *\_className* member variable of the object.
- *defaultLocale<sub>in</sub>* is the ZIL\_LOCALE\_ELEMENT class that contains the default locale information for the object.

## ZIL\_LOCALE\_MANAGER::UseLocale

### Syntax

```
#include <ui_gen.hpp>

static ZIL_LOCALE *UseLocale(const ZIL_LOCALE *locale);
    or
static const ZIL_LOCALE *UseLocale(const ZIL_ICHAR *localeName =
    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

These overloaded functions mark a ZIL\_LOCALE object as used by incrementing its *useCount* member. Whenever a library object requests the use of a ZIL\_LOCALE object, it marks the object as used by calling this function, which increments the object's *useCount* member. When the library object is done using the ZIL\_LOCALE object, it must release it by calling the **FreeLocale()** function.

The first overloaded function takes a pointer to the ZIL\_LOCALE object being marked as used.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_LOCALE object.
- *locale<sub>in</sub>* is a pointer to the ZIL\_LOCALE object that is to be marked as used.

The second overloaded function takes the internationalization name. If load capability is enabled (i.e., **ZIL\_LOAD** was defined when the library was compiled) this function will load the data from the **I18N.DAT** file if necessary. Otherwise, the data must have been compiled and linked into the application.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_ILOCALE object.
- *localeName<sub>in</sub>* is the two-letter ISO name identifying the locale to be used.



## CHAPTER 65 - ZIL MAP CHARS

The ZIL\_MAP\_CHARS class object is used to map characters between the OpenZinc standard character set and the "hardware" character set. The OpenZinc standard character set is Unicode, if the application is running in Unicode mode, or ISO8859-1 if not. The hardware character set is the character set in use on the system. The ZIL\_MAP\_CHARS class loads and maintains the required character map tables. The mapping provided by this class allows applications to properly interpret entered data at program execution. But this mapping also allows applications, like the Designer, to load a text file that was created using one character set and save it in another character set. At the time of this printing, map tables were available for the following character sets in Unicode mode: IBM 932, including extra tables for AT and NEC machines; Big Five; IBM 938, including an extra table for AT machines; IBM 1381, including an extra table for AT machines; IBM 949, including an extra table for AT machines; EUC JIS, including an extra table for Motif; IBM 437; IBM 737; IBM 850; IBM 852; IBM 855; IBM 857; IBM 860; IBM 861; IBM 863; IBM 865; IBM 866; IBM 869; IBM 1251; Macintosh; and NEXTSTEP. Map tables were available for the following character sets in ISO8859-1 mode (i.e., non-Unicode mode): IBM 437; IBM 737; IBM 850; IBM 852; IBM 855; IBM 857; IBM 860; IBM 861; IBM 863; IBM 865; IBM 866; IBM 869; IBM 1251; Macintosh; and NEXTSTEP. The complete list of available tables can be determined using the Browse utility and inspecting the **I18N.DAT** file.

The ZIL\_MAP\_CHARS class is declared in **UI\_GEN.HPP**. Its public and protected members are:

```
class ZIL_EXPORT_CLASS ZIL_MAP_CHARS : public ZIL_I18N
{
public:
    ZIL_MAP_CHARS(const ZIL_ICHAR *_mapName, const ZIL_ICHAR *extraName);
    ZIL_MAP_CHARS(const ZIL_ICHAR *_mapName, const void *fromStandard,
        const void *toStandard, int _doDelete);
    ~ZIL_MAP_CHARS();
    char *MapChar(ZIL_ICHAR mapped);
    char *MapText(const ZIL_ICHAR *mapped,
        char *unMapped = ZIL_NULLP(char), int allocate = TRUE);
    ZIL_ICHAR UnMapChar(const char *unMapped);
    ZIL_ICHAR *UnMapText(const char *unMapped,
        ZIL_ICHAR *mapped = ZIL_NULLP(ZIL_ICHAR), int allocate = TRUE);

#if defined(ZIL_UNICODE)
    int mblen(const char *hardware);
    int wcstombs(char *s, const ZIL_ICHAR *pwcs, int n = -1);
    int mbstowcs(ZIL_ICHAR *pwcs, const char *s, int n = -1);
#endif
    ZIL_ICHAR dirSepStr[2];
    int error; // Returned error of storage
    ZIL_ICHAR name[12];
};
```

## General Members

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

- *dirSepStr* is the character used to separate directories in the environment on which the application is running. For example, on Posix systems, directories are separated with the '/' character, but on Japanese systems, directories are separated with the '¥' character.
- *error* indicates if there were any errors when accessing the storage file where the map table is located.
- *name* is the name of the map table.

## ZIL\_MAP\_CHARS::ZIL\_MAP\_CHARS

### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_MAP_CHARS(const ZIL_ICHAR *_mapName, const ZIL_ICHAR *extraName);
```

*or*

```
ZIL_MAP_CHARS(const ZIL_ICHAR *_mapName, const void *fromStandard,  
const void *toStandard, int _doDelete);
```

### 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 ZIL\_MAP\_CHARS class object.

The first overloaded constructor loads the map tables from the **I18N.DAT** file.

- *\_mapName<sub>in</sub>* identifies the map table to be loaded based on the character set used by the environment. For example, in Windows NT *\_mapName* is "Unicode" since Windows NT uses the Unicode character set. In OS/2, however, *\_mapName* may be "IBM\_850" since OS/2 commonly uses code page 850.
- *extraName<sub>in</sub>* is the name of the map table that contains exceptions to the map table identified by *\_mapName*. Some character sets may be arranged slightly differently on different operating systems. This map table contains mappings for those characters that are known to be in different locations. It also contains some characters that OpenZinc uses that are not part of the normal character sets, such as text mode line draw characters.

The second overloaded constructor uses pointers to existing map tables.

- *\_mapName<sub>in</sub>* identifies the map table based on the character set used by the environment. For example, in Windows NT *\_mapName* is "Unicode" since Windows NT uses the Unicode character set. In OS/2, however, *\_mapName* may be "IBM\_850" since OS/2 commonly uses code page 850.
- *fromStandard<sub>in</sub>* is a map table that provides character mappings from the OpenZinc standard character set (either Unicode or IS08859-1) to the hardware character set.
- *toStandard<sub>in</sub>* is a map table that provides character mappings from the hardware character set to the OpenZinc standard character set. This map table will not be used in Unicode mode since *the fromStandard* map table is small enough that a simple search can provide the mapping from the hardware character set to Unicode.
- *\_doDelete<sub>in</sub>* specifies if the map tables can be deleted.

## ZIL MAP CHARS::~ZIL\_MAP CHARS

### Syntax

```
#include <ui_gen.hpp>

virtual ~ZIL_MAP_CHARS(void);
```



## Portability

This function is available on the following environments:

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

## Remarks

This virtual destructor destroys the class information associated with the ZIL\_MAP\_CHARS object.

## ZIL\_MAP\_CHARS::MapChar

### Syntax

```
#include <ui_gen.hpp>

char *MapChar(ZIL_ICHAR mapped);
```

## Portability

This function is available on the following environments:

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

## Remarks

This function maps a character from the OpenZinc standard character set to the hardware character set. **MapChar** uses static space for the return string, so its value should be copied, and should not be deleted.

- *returnValue<sub>out</sub>* is a pointer to the hardware string.
- *mapped<sub>in</sub>* is the OpenZinc standard character that is to be mapped to the hardware character set.

## ZIL\_MAP\_CHARS::MapText

### Syntax

```
#include <ui_gen.hpp>

char *MapText(const ZIL_ICHAR *mapped, char *unmapped = ZIL_NULLP(char),
              int allocate = 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 maps a string from the OpenZinc standard character set to the hardware character set. **MapText** may allocate space for the resulting string which the programmer would be responsible for deleting.

- *returnValue<sub>out</sub>* is a pointer to the hardware string.
- *mapped<sub>in</sub>* is the OpenZinc standard character string that is to be mapped to the hardware character set.
- *unmapped<sub>out</sub>* is a buffer in which the hardware string will be placed. If used, this buffer must be large enough to contain the string. If no buffer is passed in, the function can be directed to allocate a buffer or to use a temporary buffer.
- *allocate<sub>in</sub>* specifies if the function should allocate a buffer for the hardware string. If *allocate* is TRUE and no buffer was passed through the *unmapped* argument, a new buffer is allocated which the programmer is responsible for freeing. Otherwise, a static buffer is used.

## ZIL\_MAP\_CHARS::mblen

### Syntax

```
#include <ui_gen.hpp>

int mblen(const char *hardware);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function calculates how many bytes long a character is. The characters in some character sets may be from one to four bytes long. This function is used to determine the length of the first character in a string of multi-byte characters. This function is defined in Unicode mode only.

- *returnValue*<sub>out</sub> is the number of bytes in the first character of the string passed in.
- *hardware*<sub>in</sub> is the multi-byte character string of which the first character's size is required.

## ZIL\_MAP\_CHARS::mbstowcs

### Syntax

```
#include <ui_gen.hpp>

int mbstowcs(ZIL_ICHAR *pwcs, const char *s, int n = -1);
```

## Portability

This function is available on the following environments:

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

## Remarks

This function converts a multi-byte character string to a wide-character Unicode string. The source string is made up of characters from any hardware character set. This function is defined in Unicode mode only.

- *returnValue* is a count of how many characters are in the converted string.
- *pwcs<sub>out</sub>* is a pointer to a buffer in which the converted wide-character Unicode string will be placed. This buffer must be large enough to contain the string.
- *s<sub>in</sub>* is the source string to be converted. This string is made up of characters from the local hardware character set.
- *n<sub>in</sub>* is a count of how many characters are to be converted. If *n* is less than 0, the `strlen` of *s* is used, **strlen** will provide the maximum number of characters that the input string may contain since **strlen** will not necessarily return the number of actual characters in the string, but rather the number of 8-bit values in the string. Some characters may be more than 8-bits wide.

## ZIL\_MAP\_CHARS::UnMapChar

### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_ICHAR UnMapChar(const char *unMapped);
```

## Portability

This function is available on the following environments:

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

## Remarks

This function unmaps a character from the hardware character set and returns the equivalent character from the OpenZinc standard character set.

- *returnValue<sub>out</sub>* is the unmapped OpenZinc standard character.
- *unMapped<sub>in</sub>* is a pointer to a buffer containing the hardware character.

## ZIL\_MAP\_CHARS::UnMapText

### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_ICHAR *UnMapText(const char *unmapped,  
ZIL_ICHAR *mapped = ZIL_NULLP(ZIL_ICHAR), int allocate = 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 unmaps a string from a hardware character set to a OpenZinc standard character string.

- *returnValue<sub>out</sub>* is a pointer to the unmapped OpenZinc standard text.
- *unmapped<sub>in</sub>* is a pointer to a buffer containing the mapped hardware text.

- *mapped<sub>out</sub>* is a pointer to a buffer in which the unmapped OpenZinc standard text will be placed.
- *allocate<sub>in</sub>* indicates if the function should allocate a buffer for the unmapped OpenZinc standard text. If *allocate* is TRUE and no buffer is passed through the *mapped* argument, the function will allocate a buffer. The programmer is responsible for deleting the buffer. If *allocate* is FALSE, a static buffer is used.

## ZIL\_MAP\_CHARS::wcstombs

### Syntax

```
#include <ui_gen.hpp>
```

```
int wcstombs(char *s, const ZIL_ICHAR *pwcs, int n = -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 converts a wide-character Unicode string to a multi-byte character string. The destination string will be made up of characters from the hardware character set. This function is defined in Unicode mode only.

- *returnValue* is a count of how many characters are in the converted string.
- *s<sub>out</sub>* is a pointer to a buffer in which the multi-byte character string will be placed after it has been converted from the Unicode string. This string will be made up of characters from the local hardware character set. The buffer must be large enough to hold the converted string.
- *pwcs<sub>in</sub>* is a pointer to the wide-character Unicode string.
- *n<sub>in</sub>* is a count of how many characters are to be converted. If *n* is less than 0, the **strlen** of 5 is used.



## CHAPTER 66 - ZIL\_STORAGE

The ZIL\_STORAGE class is used to write OpenZinc Application Framework data files. It is created as a class so that the file can be treated as an object, which does the writing. Because ZIL\_STORAGE is derived from ZIL\_STORAGE\_READ\_ONLY, this class inherits reading functionality.

Although the ZIL\_STORAGE is stored in a file, it should be thought of as a file system and not just a file. The ZIL\_STORAGE class is similar, in design, to the Unix file system. This means that within a ZIL\_STORAGE object, there can be many files (i.e., objects derived from ZIL\_STORAGE\_OBJECT) and levels of sub-directories. The programmer has the ability to copy, delete and move files across directories. The maximum length of a ZIL\_STORAGE object is about 16 megabytes with the maximum size of an individual object being 4 megabytes. A single ZIL\_STORAGE object may contain a maximum of 16,000 objects (i.e., ZIL\_STORAGE\_OBJECT.)

The ZIL\_STORAGE class is typically used for persistent objects (e.g., objects created using OpenZinc Designer) and for storing internationalization data. It is also commonly used as a simple data base.

The ZIL\_STORAGE class is declared in UI\_GEN.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS ZIL_STORAGE : public ZIL_STORAGE_READ_ONLY
{
public:
    // Read/Write Storage
    ZIL_STORAGE(void);
    ZIL_STORAGE(const ZIL_ICHAR *name, UIS_FLAGS pFlags = UIS_READWRITE);
    ~ZIL_STORAGE(void);
    int DestroyObject(const ZIL_ICHAR *name);
    int Flush(void);
    int Link(const ZIL_ICHAR *path1, const ZIL_ICHAR *path2);
    int Mkdir(const ZIL_ICHAR *newName);
    int RenameObject(const ZIL_ICHAR *oldObject, const ZIL_ICHAR *newName);
    int Rmdir(const ZIL_ICHAR *name);
    int Save(int revisions = 0);
    int SaveAs(const ZIL_ICHAR *newName, int revisions = 0);
};
```

### General Members

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



## ZIL\_STORAGE::ZIL\_STORAGE

### Syntax

```
#include <ui_gen.hpp>

ZIL_STORAGE(void);
    or
ZIL_STORAGE(const ZIL_ICHAR *name, UIS_FLAGS pFlags = UIS_READWRITE);
```

### 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 both create a new `ZIL_STORAGE` class object.

The first overloaded constructor creates a `ZIL_STORAGE` with no associated disk file.

The second overloaded constructor creates a `ZIL_STORAGE` and binds it to a disk file.

- *name<sub>in</sub>* is the name of the file to be opened, *name* can include a path, if desired. If the storage file cannot be opened, this function will use `UI_PATH` functions to search other paths for the file. For example, "`~zinc~examples~*.dat`" will return the first `.DAT` file found with that path. Notice that the `~` character is used as the directory separator. The `*` character is considered to be a wildcard character.
- *pFlags<sub>in</sub>* indicates how a file is to be opened. The following `UIS_FLAGS` are supported:

`UIS_READ`—Opens the file for read access only.

`UIS_READWRITE`—Opens the file for read and write access. This flag allows modifications to be made to the file.

`UIS_CREATE`—Creates and opens a file for write access. Any previous file will be deleted.

**UIS\_OPENCREATE**—Opens an existing object for read and write access. If the object does not exist, it is created for read and write access.

**UIS\_TEMPORARY**—Creates the file as a temporary file. When ZIL\_STORAGE is destroyed, the file will be deleted.

## Example

```
#include <ui_win.hpp>

UI_HELP_SYSTEM::UI_HELP_SYSTEM(char *pathName,
    UI_WINDOW_MANAGER *windowManager, UI_HELP_CONTEXT helpContext) :
    defaultHelpContext(helpContext)
{
    // Open the help storage unit.
    storage = new ZIL_STORAGE(pathName, UIS_READ);
    if (storage->storageError)
    {
        delete storage;
        storage = NULL;
    }
}

}
```

## ZIL\_STORAGE::~~ZIL\_STORAGE

### Syntax

```
#include <ui_gen.hpp>

~ZIL_STORAGE(void);
```

### Portability

This function is available on the following environments:

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

### Remarks

This destructor destroys the class information associated with the ZIL\_STORAGE object and closes any files opened by the ZIL\_STORAGE constructor. If a file was opened with the UIS\_TEMPORARY flag, it will be deleted when this destructor is called.

## Example

```
#include <ui_win.hpp>

UI_HELP_SYSTEM::~UI_HELP_SYSTEM(void)
{
    if (storage)
        delete storage;
    delete helpWindow;
}
```

## ZIL\_STORAGE::Destroy Object

### Syntax

```
#include <ui_gen.hpp>

int DestroyObject(const ZIL_ICHAR *name)
```

### Portability

This function is available on the following environments:

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

### Remarks

This removes an object from the ZIL\_STORAGE file. After it is removed, the object is destroyed. The file space associated with the object is not freed, but is re-used if a new object is written to the file.

- *returnValue*<sub>out</sub> is 0 on success. If a failure occurred, -1 is returned.
- *name*<sub>in</sub> is the name of the object to be destroyed.

## Example

```
tinclude <ui_win.hpp>

int RemoveDirectory(ZIL_STORAGE *storage, const char *name,
const char *directoryName)
{
    // Clean the directory.
    for (name = FindFirstObject(name); name; name = FindFirstObject(name))
        DestroyObject(name);
}
```

```

        storage->ChDir("..");
        storage->RmDir(directoryName);
    }

```

## **ZIL\_STORAGE::Flush**

### **Syntax**

```

#include <ui_gen.hpp>

int Flush(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 writes the internal cache buffer to a temporary file. One use of this function would be to save backup copies of a file within a timer function. **Flush()** does not update the actual storage file. To commit the changes to the storage file, use **Save()**.

- *returnValue<sub>out</sub>* is 0 on success and -1 if an error occurs.

### **Example**

```

#include <ui_gen.hpp>

int TimerBackup(ZIL_STORAGE *storage)
{
    storage->Flush();
}

```

## **ZIL STORAGE::Link**

### **Syntax**

```
#include <ui_gen.hpp>
```

```
int Link(const ZIL_ICHAR *path1, const ZIL_ICHAR *path2)\
```

### **Portability**

This function is available on the following environments:

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

### **Remarks**

This function gives an existing object a second name. This allows a second (or later) instance of an object within a .DAT file to reference, or point to, the data stored with the original instance of the object. Thus, many objects can use the same data without duplicating the data for each instance. After this function is called, both names refer to the same object. Deleting one name will not delete the other name or the object. This is different than a rename.

- *returnValue*<sub>out</sub> is 0 on success and -1 if an error occurs.
- *path1*<sub>in</sub> is the original name (including its path) of the object .
- *path2*<sub>in</sub> is the new name (including its path) by which the object can also be referenced.

### **Example**

```
#include <ui_gen.hpp>

int EmployeeSetup(void)

{
    storage->Link("-employees-Blake", "-development-Blake");
}
```

## ZIL\_STORAGE::MkDir

### Syntax

```
#include <ui_gen.hpp>

int MkDir(const ZIL_ICHAR *newName);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function makes a new directory within a ZIL\_STORAGE file.

- *returnValue<sub>out</sub>* is 0 on success and -1 on failure.
- *newName<sub>in</sub>* is the name of the directory to be created. For example, "~UI\_HPP~MYDIR" can be specified to make the directory "MYDIR" in the UI\_HPP subdirectory.

### Example

```
#include <ui_gen.hpp>

main(int argc, char *argv[])
{

    // Create the help directory.
    ZIL_STORAGE *helpFile = new ZIL_STORAGE(fileName, UIS_READWRITE);
    if (helpFile->storageError)
    {
        delete helpFile;
        helpFile = new ZIL_STORAGE(fileName, UIS_CREATE | UIS_READWRITE);
    }
    helpFile->MkDir("UI_HPP");
    if (newHelpDirectory)
        helpFile->Rmdir("UI_HELP_CONTEXT");
    helpFile->MkDir("UI_HELP_CONTEXT");

    // Print genhelp status.
    PrintStatus("PROCESSING %s:\n", fileName);
```

```

ZIL_OBJECTID helpID = 0;

// Generate the HPP directory.
helpFile->ChDir("~UI_HPP");
ZIL_STORAGE_OBJECT *hppElement = new ZIL_STORAGE_OBJECT(*helpFile,
    "HELP_CONTEXTS", ID_HELP_CONTEXT, UIS_CREATE | UIS_READWRITE);

// Generate the help contexts.
helpFile->ChDir("~UI_HELP_CONTEXT");

}

```

## ZIL\_STORAGE::RenameObject

### Syntax

```
#include <ui_gen.hpp>
```

```
int RenameObject(const ZIL_ICHAR *oldObject, const ZIL_ICHAR *newName);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function changes the name (i.e., *stringID*) of an object inside the storage object.

- *returnValue<sub>out</sub>* is 0 on success and -1 on failure.
- *oldObject<sub>ini</sub>* is the name of the object whose name is to be changed.
- *newName<sub>in</sub>* is the new name of the object.

### Example

```

#include <ui_gen.hpp>

ExampleFunction(ZIL_STORAGE *storage)
{

```

```
        storage->RenameObject("Item1", "FirstItem");
    }
```

## **ZIL\_STORAGE::RmDir**

### **Syntax**

```
#include <ui_gen.hpp>

int RmDir(const ZIL_ICHAR *name);
```

### **Portability**

This function is available on the following environments:

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

### **Remarks**

This function removes a directory within a ZIL\_STORAGE file.

- *returnValue<sub>out</sub>* is 0 on success and -1 on failure.
- *name<sub>in</sub>* is the name of the directory to be removed.

NOTE: A directory must be empty in order for it to be deleted.

### **Example**

```
#include <ui_win.hpp>

main(int argc, char *argv[])

{

    // Create the help directory.
    ZIL_STORAGE *helpFile = new ZIL_STORAGE(fileName, UIS_READWRITE);
    if ~(helpFile->storageError)
    {
        delete helpFile;
        helpFile = new ZIL_STORAGE (fileName, UIS_CREATE I UIS_READWRITE);
    }
}
```



```

}
helpFile->Mkdir("UI_HPP");
if (newHelpDirectory)
    helpFile->Rmdir("UI_HELP_CONTEXT");
helpFile->Mkdir("UI_HELP_CONTEXT");

// Print genhelp status.
PrintStatus("PROCESSING %s:\n", fileName);
ZIL_OBJECTID helpID = 0;

// Generate the HPP directory.
helpFile->ChDir("~UI_HPP");
ZIL_STORAGE_OBJECT *hppElement = new ZIL_STORAGE_OBJECT(*helpFile,
    "HELP_CONTEXTS", ID_HELP_CONTEXT, UIS_CREATE | UIS_READWRITE);

// Generate the help contexts.
helpFile->ChDir ("~UI_HELP_CONTEXT");

```

## ZIL\_STORAGE::Save

### Syntax

```

#include <ui_gen.hpp>

int Save(int revisions = 0);

```

### Portability

This function is available on the following environments:

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

### Remarks

This function saves the storage file and all of the information contained in it. This operation (or a call to SaveAs()) must be performed even if each individual object is stored. Since some objects are saved in temporary files, this is the only way to ensure that the information is saved to the main storage file.

- *returnValue*<sub>out</sub> is 0 on success, or -1 on failure.

- *revisions<sub>m</sub>* is the number of backup files to be kept. Backup files are files with the .BK? extension where the "?" denotes which backup number the file is. For example, **TEST.BK1** would be the most recent backup of the file **TEST.DAT**, **TEST.BK2** would be the previous backup of the file **TEST.DAT**, etc.

**NOTE:** A backup file is only created the first time **Save()** is called. The different backup file revisions are created from previous times the file was opened. To create another backup file, you must close and then re-open the storage.

### Example

```
#include <ui_gen.hpp>

int ZIL_STORAGE::SaveAs(const char *newName, int revisions)
{
    if (!FlagSet (flags, UIS_READWRITE) )
        storageError = EACCES;
        return -1;
    }
    if (modified) (void) time (&sb->modifytime) ;
    Flush();
    StripFullPath(newName, pname, fname);
    firstTime = 1;
    Save(revisions);
    return 0;
}
```

## ZIL\_STORAGE::SaveAs

### Syntax

```
#include <ui_gen.hpp>

int SaveAs(ZIL_ICHAR *newName, int revisions = 0);
```

### Portability

This function is available on the following environments:

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

## Remarks

This function saves the storage file and all of the information contained in it. This operation (or a call to **Save()**) must be performed even if each individual object is stored. Since some objects are saved in temporary files, this is the only way to ensure that the information is saved to the main storage file.

- *returnValue<sub>out</sub>* is 0 on success, or -1 on failure.
- *newName<sub>in</sub>* is the new name of the storage file.
- *revisions<sub>in</sub>* is the number of backup files to be kept. Backup files are files with the .BK? extension where the "?" denotes which backup number the file is. For example, **TEST.BK1** would be the most recent backup of the file **TEST.DAT**, **TEST.BK2** would be the previous backup of the file **TEST.DAT**, etc.

**NOTE:** A backup file is only created the first time **SaveAs( )** is called. The different backup file revisions are created from previous times the file was opened. To create another backup file, you must close and then re-open the storage.

## Example

```
#include <ui_gen.hpp>

int CloseAll(ZIL_STORAGE *storage, const char *newName, int revisions)
{
    char currentName[129];

    Flush();
    StorageName(currentName);
    if (!strcmp(currentName, newName));
        Save(revisions);
    else
        SaveAs(newName, revisions);
    return 0;
}
```

## CHAPTER 67 - ZIL\_STORAGE\_DIRECTORY

The ZIL\_STORAGE\_DIRECTORY class object provides a directory pointer and the functionality to manipulate the directory pointer. It is used by ZIL\_STORAGE\_READ\_ONLY and ZIL\_STORAGE to maintain multiple directory pointers. A new instance of ZIL\_STORAGE\_DIRECTORY can only be created using the **ZIL\_STORAGE\_READ\_ONLY::OpenDir( )** function. This class is typically not used by the programmer.

The ZIL\_STORAGE\_DIRECTORY class is declared in **UI\_GEN.HPP**. Its public and protected members are:

```
Class ZIL_EXPORT_CLASS ZIL_STORAGE_DIRECTORY
{
public:
    ~ZIL_STORAGE_DIRECTORY(void);
    directoryEntry *ReadDir(void);
    void RewindDir(void);
    void SeekDir(ZIL_UINT16 _position);
    ZIL_UINT16 TellDir(void);
};
```

### General Members

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

### ZIL\_STORAGE\_DIRECTORY::~ZIL\_STORAGE\_DIRECTORY

#### Syntax

```
#include <ui_gen.hpp>
```

```
virtual ~ZIL_STORAGE_DIRECTORY(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 ZIL\_STORAGE\_DIRECTORY object.

## ZIL\_STORAGE\_DIRECTORY::ReadDir

### Syntax

```
#include <ui_gen.hpp>
```

```
directoryEntry *ReadDir(void);
```

### 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 obtains information about the current directory.

- *returnValue<sub>out</sub>* is a pointer to **directoryEntry**, a structure that contains some data about the directory.

## ZIL\_STORAGE\_DIRECTORY::RewindDir

### Syntax

```
#include <ui_gen.hpp>
```

```
void RewindDir(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 repositions the directory file pointer to the position it was at when the ZIL\_STORAGE\_DIRECTORY object was created.

## ZIL\_STORAGE\_DIRECTORY::SeekDir

### Syntax

```
#include <ui_gen.hpp>

void SeekDir(ZIL_UINT 16 _position);
```

## 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 positions the file pointer to the position specified by *\_position*. This function should only be used with a value obtained from **TellDir( )**.

- *\_position<sub>in</sub>* is the position to which the file pointer should be set.

## ZIL\_STORAGE\_DIRECTORY::TellDir

### Syntax

```
#include <ui_gen.hpp>

ZIL_UINT16 TellDir(void);
```

### 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 returns the current position of the file pointer. This position should only be used for setting the position with the **SeekDir()** function.

- *returnValue<sub>out</sub>* is the current position of the file pointer.

## CHAPTER 68 - ZIL\_STORAGE\_OBJECT

The ZIL\_STORAGE\_OBJECT class is used to store data in OpenZinc Application Framework data files. The most common use is to store persistent objects (e.g., objects created using OpenZinc Designer) using OpenZinc data files created with ZIL\_STORAGE. Although the ZIL\_STORAGE\_OBJECT is stored in a file, it should be thought of as a file and not a record in a file. A ZIL\_STORAGE\_OBJECT may have many pieces of data associated with it. As each piece of data is stored to, or loaded from, the ZIL\_STORAGE\_OBJECT its file pointer is advanced so that it is pointing to the next piece of data. Because ZIL\_STORAGE\_OBJECT is derived from ZIL\_STORAGE\_OBJECT\_READ\_ONLY, this class can also read from the ZIL\_STORAGE file. See "Chapter 66—ZIL\_STORAGE" in this manual for more information regarding data files.

The ZIL\_STORAGE\_OBJECT class is declared in **UI\_GEN.HPP**. Its public and protected members are:

```
class ZIL_EXPORT CLASS ZIL_STORAGE_OBJECT : public
    ZIL_STORAGE_OBJECT_READ_ONLY
{
public:
    // Read/Write support
    ZIL_STORAGE_OBJECT(void);
    ZIL_STORAGE_OBJECT(ZIL_STORAGE &file, const ZIL_ICHAR *name,
        OBJECTID nObjectID, UIS_FLAGS pFlags = UIS_READWRITE);
    ~ZIL_STORAGE_OBJECT(void);

    void SetCTime(ZIL_INT32 val);
    void SetMTime(ZIL_INT32 val);
    void Touch(void);
    virtual int Store(ZIL_INT16 value);
    virtual int Store(ZIL_UINT16 value);
    virtual int Store(ZIL_INT32 value);
    virtual int Store(ZIL_UINT32 value);
    virtual int Store(ZIL_INT8 value);
    virtual int Store(ZIL_UINT8 value);
    virtual int Store(void *buff, int size, int length);
    virtual int Store(const ZIL_ICHAR *string);
};
```

### General Members

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



## **ZIL STORAGE OBJECT::ZIL STORAGE OBJECT**

### **Syntax**

```
#include <ui_gen.hpp>

ZIL_STORAGE_OBJECT(void);
    or
ZIL_STORAGE_OBJECT(ZIL_STORAGE &file, const ZIL_ICHAR *name,
    ZIL_OBJECTID nObjectID, UIS_FLAGS pFlags = UIS_READWRITE);
```

### **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 each create a new `ZIL_STORAGE_OBJECT`.

The first overloaded constructor creates a `ZIL_STORAGE_OBJECT` with no associated object. This advanced constructor is reserved for internal use only.

The second overloaded constructor creates a `ZIL_STORAGE_OBJECT` with the following parameters:

- *file<sub>in</sub>* is the file containing the object. If the object is not found, it will be created if `UIS_CREATE` or `UIS_OPENCREATE` is specified.
- *name<sub>in</sub>* is the name of the object.
- *nObjectID<sub>in</sub>* is the *objectID* of the object.
- *pFlags<sub>in</sub>* indicates how the storage object is to be opened. The following `UIS_FLAGS` are supported:

**UIS\_READ**—Allows read only access to the object.

**UIS\_READWRITE**—Allows read and write access to the object. This flag allows modifications to be made to the object.

**UIS\_CREATE**—Creates an object and allows write access to it. Any previous object will be deleted.

**UIS\_OPENCREATE**—Opens an existing object for read and write access. If the object does not exist, it is created for read and write access.

## Example

```
#include <ui_win.hpp>

void UIW_WINDOW::Load(const char *name, ZIL_STORAGE *directory,
ZIL_STORAGE_OBJECT *file)
{
    // Check for a valid directory and file,
    int tempDirectory = FALSE, tempFile = FALSE;
    if (name && !file)
    {
        char pathName[128], fileName[32], objectName[32];
        ZIL_STORAGE::StripFullPath(name, pathName, fileName, objectName);
        if (!directory)
        {
            ZIL_STORAGE::AppendFullPath(pathName, pathName, fileName);
            ZIL_STORAGE::ChangeExtension(pathName, ".dat");
            directory = new ZIL_STORAGE(pathName, UIS_READ);
            tempDirectory = TRUE;
        }
        if (!file)
        {
            if (objectName[0] == '\0')
                strcpy(objectName, fileName);
            directory->ChDir("~UIW_WINDOW");
            file = new ZIL_STORAGE_OBJECT('directory', objectName, ID_WINDOW,
UIS_READ);
            if (file->objectError)
        }
    }
}
```

## **ZIL STORAGE OBJECT::~ZIL STORAGE OBJECT**

### **Syntax**

```
#include <ui_gen.hpp>

~ZIL_STORAGE_OBJECT(void);
```

### **Portability**

This function is available on the following environments:

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

### **Remarks**

This destructor destroys the class information associated with the ZIL\_STORAGE\_OBJECT.

### **Example**

```
#include <ui_win.hpp>

void UIW_WINDOW::Load(const char *name, ZIL_STORAGE *directory,
    ZIL_STORAGE_OBJECT *file)
{

    // Clean up the file and storage,
    if (tempFile)
        delete file;
    if (tempDirectory)
        delete directory;
}
```

## **ZIL STORAGE OBJECT::SetCTime**

### **Syntax**

```
#include <ui_gen.hpp>

void SetCTime(ZIL_INT32 val);
```

## 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 create time for the object.

- *val<sub>in</sub>* is the new create time.

## ZIL\_STORAGE\_OBJECT::SetMTime

### Syntax

```
#include <ui_gen.hpp>

void SetMTime(ZIL_INT32 val);
```

## 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 modify time for the object.

- *val<sub>in</sub>* is the new modify time.

## **ZIL STORAGE OBJECT::Store**

### **Syntax**

```
#include <ui_gen.hpp>

int Store(ZIL_INT16 value);
    or
int Store(ZIL_UINT16 value);
    or
int Store(ZIL_INT32 value);
    or
int Store(ZIL_UINT32 value);
    or
int Store(ZIL_UINT8 value);
    or
int Store(ZIL_INT8 value);
    or
int Store(void *buff, int size, int length);
    or
int Store(const ZIL_ICHAR *string);
```

### **Portability**

These functions are available on the following environments:

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

### **Remarks**

The first six overloaded functions write information to the storage file according to the type of value given.

- *returnValue<sub>out</sub>* is the number of bytes written.
- *value<sub>in</sub>* is the numeric value to be written. The following values are supported:
  - ZIL\_INT8**—A number whose value is between -128 and 127 (8 bits, signed).
  - ZIL\_UINT8**—A number whose value is between 0 and 255 (8 bits, unsigned).

**ZIL\_INT16**—A number whose value is between -32,768 and 32,767 (16 bits, signed).

**ZIL\_UINT16**—A number whose value is between 0 and 65,535 (16 bits, unsigned).

**ZIL\_INT32**—A number whose value is between -2,147,483,648 and 2,147,483,647 (32 bits, signed).

**ZIL\_UINT32**—A number whose value is between 0 and 4,294,967,295 (32 bits, unsigned).

The seventh overloaded function writes information into the storage file according to the following values:

- *returnValue<sub>out</sub>* is the number of bytes written.
- *buff<sub>in</sub>* is a pointer to the buffer that contains the information to be written.
- *size<sub>in</sub>* is the size of each item to be written.
- *length<sub>in</sub>* is the number of items to be written.

In general, programmers are discouraged from using this function, because the integrity of the type of value being stored cannot be guaranteed across environments. For example, the storage size of a value (e.g., int) in DOS might be different than that in Motif. All of the other **Store()** functions, however, are the same across environments.

The eighth overloaded function writes information into the storage file according to the following value:

- *returnValue<sub>out</sub>* is the number of bytes written.
- *string<sub>in</sub>* is a pointer to the string that is to be written.

## ZIL\_STORAGE\_OBJECT::Touch

### Syntax

```
#include <ui_gen.hpp>

void Touch(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 sets *ZIL\_STORAGE::inode.modifyTime* (the time of the last modification) of the storage object to the current time.

### Example

```
#include <ui_gen.hpp>

ExampleFunction(ZIL_STORAGE_OBJECT *object)
{
    object->Touch();
}
```

## CHAPTER 69 - ZIL STORAGE OBJECT READ ONLY

The `ZIL_STORAGE_OBJECT_READ_ONLY` class is used to load an object's data using OpenZinc Application Framework data files. The most common use is to load persistent objects (e.g., objects created using OpenZinc Designer) using OpenZinc data files created with `ZIL_STORAGE_READ_ONLY`. Although the `ZIL_STORAGE_OBJECT_READ_ONLY` is stored in a file, it should be thought of as a file and not a record in a file. A `ZIL_STORAGE_OBJECT` may have many pieces of data associated with it. As each piece of data is loaded from the `ZIL_STORAGE_OBJECT_READ_ONLY` its file pointer is advanced so that it is pointing to the next piece of data. `ZIL_STORAGE_OBJECT` only allows data to be loaded from the file. If write access is needed use the `ZIL_STORAGE_OBJECT` class. See "Chapter 70—`ZIL_STORAGE_READ_ONLY`" in this manual for more information regarding data files.

The `ZIL_STORAGE_OBJECT_READ_ONLY` class is declared in `UI_GEN.HPP`. Its public and protected members are:

```
class ZIL_EXPORT CLASS ZIL_STORAGE_OBJECT_READ_ONLY : public
    ZIL_INTERNATIONAL
{
public:          // Read-Only support
    int objectError;
    OBJECTID objectID;
    ZIL_ICHAR *stringID;

    ZIL_STORAGE_OBJECT_READ_ONLY(void);
    ZIL_STORAGE_OBJECT_READ_ONLY(ZIL_STORAGE_READ_ONLY &file,
        const ZIL_ICHAR *name, OBJECTID nObjectID);
    virtual ~ZIL_STORAGE_OBJECT_READ_ONLY(void);

    long Seekflong _position;
    ZIL_STATS_INFO *Stats(void);
    ZIL_STORAGE_READ_ONLY *Storage(void);
    long Tell(void);
    virtual int Load(ZIL_INT16 *value);
    virtual int Load(ZIL_UINT16 *value);
    virtual int Load(ZIL_INT32 *value);
    virtual int Load(ZIL_UINT32 *value);
    virtual int Load(ZIL_UINT8 *value);
    virtual int Load(ZIL_INT8 *value);
    virtual int Load(void *buff, int size, int length);
    virtual int Load(ZIL_ICHAR *string, int length);
    virtual int Load(ZIL_ICHAR **string);

    virtual int Store(void *buff, int size, int length);
};
```



## General Members

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

- *objectError* is the result of the last attempt to load this object from a file. This value is set to one of the values defined by *errno*. For more information see the global variable *errno* in your compiler language reference manual.
- *objectID* is the *objectID* for the type of object being loaded.
- *stringID* is the name of the object.

## **ZIL\_STORAGE\_OBJECT\_READ\_ONLY::ZIL\_STORAGE\_OBJECT\_READ\_ONLY**

### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_STORAGE_OBJECT_READ_ONLY(void);
```

*or*

```
ZIL_STORAGE_OBJECT_READ_ONLY(ZIL_STORAGE_READ_ONLY &file,  
    const ZIL_ICHAR *name, ZIL_OBJECTID nObjectID);
```

### 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 each create a new `ZIL_STORAGE_OBJECT_READ_ONLY`.

The first overloaded constructor creates a `ZIL_STORAGE_OBJECT_READ_ONLY` with no associated object. This advanced constructor is reserved for internal use only.

The second overloaded constructor creates a ZIL\_STORAGE\_OBJECT\_READ\_ONLY with the following parameters:

- *file<sub>in</sub>* is the file containing the object.
- *name<sub>in</sub>* is the name of the object.
- *nObjectID<sub>in</sub>* is the *objectID* of the object.

## Example

```
#include <ui_win.hpp>

void UIW_WINDOW::Load(const char *name, ZIL_STORAGE_READ_ONLY *directory,
                     ZIL_STORAGE_OBJECT_READ_ONLY *file)
{
    // Check for a valid directory and file,
    int tempDirectory = FALSE, tempFile = FALSE;
    if (name && !file)
    {
        char pathName [128], fileName [32], objectName[32];
        ZIL_STORAGE_READ_ONLY::StripFullPath(name, pathName, fileName,
                                             objectName);
        if (!directory)
        {
            ZIL_STORAGE_READ_ONLY::AppendFullPath(pathName, pathName, fileName);
            ZIL_STORAGE_READ_ONLY::ChangeExtension(pathName, ".dat");
            directory = new ZIL_STORAGE_READ_ONLY(pathName, UIS_READ);
            tempDirectory = TRUE;
        }
        if (!file)
        {
            if (objectName[0] == '\0')
                strcpy(objectName, fileName);
            directory->ChDir("~/UIW_WINDOW");
            file = new ZIL_STORAGE_OBJECT_READ_ONLY(*directory, objectName,
                        ID_WINDOW, UIS_READ);
            if (file->objectError)
        }
    }
}
```

## **ZIL\_STORAGE\_OBJECT\_READ\_ONLY::~ZIL\_STORAGE\_OBJECT\_READ\_ONLY**

### **Syntax**

```
#include <ui_gen.hpp>

~ZIL_STORAGE_OBJECT_READ_ONLY(void);
```

### **Portability**

This function is available on the following environments:

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

### **Remarks**

This destructor destroys the class information associated with the ZIL\_STORAGE\_OBJECT\_READ\_ONLY.

### **Example**

```
#include <ui_win.hpp>

void UIW_WINDOW::Load(const char *name, ZIL_STORAGE_READ_ONLY *directory,
    ZIL_STORAGE_OBJECT_READ_ONLY *file)
{

    // Clean up the file and storage.
    if (tempFile)
        delete file;
    if (tempDirectory)
        delete directory;
}
```

## ZIL\_STORAGE\_OBJECT\_READ\_ONLY::Load

### Syntax

```
#include <ui_gen.hpp>

int Load(ZIL_INT16 *value);
    or
int Load(ZIL_UINT16 *value);
    or
int Load(ZIL_INT32 *value);
    or
int Load(ZIL_UINT32 *value);
    or
int Load(ZIL_UINT8 *value);
    or
int Load(ZIL_INT8 *value);
    or
int Load(void* buff, int size, int length);
    or
int Load(ZIL_ICHAR *string, int length);
    or

int Load(ZIL_ICHAR **string);
```

### Portability

These functions are available on the following environments:

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

### Remarks

The first six overloaded functions read information from the storage file according to the type of value given.

- *returnValue<sub>out</sub>* is the number of bytes read.
- *value<sub>out</sub>* is the numeric value read. The following values are supported:

**ZIL\_INT8**—A number whose value is between -128 and 127 (8 bits, signed).

**ZIL\_UINT8**—A number whose value is between 0 and 255 (8 bits, unsigned).

**ZIL\_INT16**—A number whose value is between -32,768 and 32,767 (16 bits, signed).

**ZIL\_UINT16**—A number whose value is between 0 and 65,535 (16 bits, unsigned).

**ZIL\_INT32**—A number whose value is between -2,147,483,648 and 2,147,483,647 (32 bits, signed).

**ZIL\_UINT32**—A number whose value is between 0 and 4,294,967,295 (32 bits, unsigned).

The seventh overloaded function reads information from the storage file according to the following values:

- *buff<sub>out</sub>* is a pointer to the buffer that will receive the information. This buffer must be large enough to contain the information read.
- *size<sub>in</sub>* is the size of each item to be read.
- *length<sub>in</sub>* is the number of items to be read.

In general, programmers are discouraged from using this function, because the integrity of the type of value being loaded cannot be guaranteed across environments. For example, the storage size of a value in DOS might be different than that in Motif. All of the other **Load( )** functions, however, are the same across environments.

The eighth overloaded function reads information from the storage file according to the following values:

- *string<sub>out</sub>* is a pointer to the character buffer that will receive the information. This buffer must be large enough to contain the information read.
- *length<sub>in</sub>* is the number of characters to read.

The ninth overloaded function reads information from the storage file according to the following values:

- *string<sub>out</sub>* is a pointer to a string pointer where the information will be written. This string is allocated by the library.

## **ZIL STORAGE OBJECT READ ONLY::Seek**

### **Syntax**

```
#include <ui_gen.hpp>

long Seek(long _position);
```

### **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 positions the file pointer to the position specified by *\_position*. This function should only be used with a value obtained from Tell( ).

- *returnValue<sub>out</sub>* is the updated file position.
- *\_position<sub>in</sub>* is the position to which the file pointer should be set.

## **ZIL STORAGE OBJECT READ ONLY::Stats**

### **Syntax**

```
#include <ui_gen.hpp>

ZIL_STATS_INFO *Stats(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 some statistics regarding the ZIL\_STORAGE\_OBJECT\_READ\_ONLY.

- *returnValue<sub>out</sub>* is a pointer to a ZIL\_STATS\_INFO structure. For more information regarding ZIL\_STATS\_INFO, see the beginning of "Chapter 70—ZIL\_STORAGE\_READ\_ONLY." If an error occurs, NULL is returned.

## **ZIL\_STORAGE\_OBJECT\_READ\_ONLY::Storage**

### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_STORAGE_READ_ONLY *Storage(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 ZIL\_STORAGE\_READ\_ONLY that contains the ZIL\_STORAGE\_OBJECT\_READ\_ONLY.

- *returnValue<sub>out</sub>* is a pointer to a ZIL\_STORAGE\_READ\_ONLY file.

## **ZIL\_STORAGE\_OBJECT\_READ\_ONLY::Store**

### Syntax

```
#include <ui_gen.hpp>
```

```
virtual int Store(void *buff, int size, int length);
```

## 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 allows the programmer to have low-level access to the file.

This function writes information into the storage file according to the following values. In general, programmers are discouraged from using this function, because the integrity of the type of value being loaded cannot be guaranteed across environments. For example, the storage size of a value (e.g., int) in DOS might be different than that in Motif. All of the other **Store( )** functions, however, are the same for DOS and Motif.

- *buff<sub>in/out</sub>* is a pointer to the buffer that contains the information to be written.
- *size<sub>in</sub>* is the size of each item to be written.
- *length<sub>in</sub>* is the number of items to be written.

## ZIL\_STORAGE OBJECT READ ONLY::Tell

### Syntax

```
#include <ui_gen.hpp>
```

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

### 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 returns the current position of the file pointer.

- *returnValue<sub>out</sub>* is the current position of the file pointer.

## CHAPTER 70 - ZIL\_STORAGE\_READ\_ONLY

The ZIL\_STORAGE\_READ\_ONLY class is used to read OpenZinc Application Framework data files. It is created as a class so that the file can be treated as an object, which does the reading.

Although the ZIL\_STORAGE\_READ\_ONLY is stored in a file, it should be thought of as a file system and not just a file. The ZIL\_STORAGE\_READ\_ONLY class is similar, in design, to the Unix file system. This means that within a ZIL\_STORAGE\_READ\_ONLY object, there can be many files (i.e., objects derived from ZIL\_STORAGE\_READ\_ONLY\_OBJECT) and levels of sub-directories. The programmer has the ability to copy, delete and move files across directories. The maximum length of a ZIL\_STORAGE\_READ\_ONLY object is about 16 megabytes with the maximum size of an individual object being 4 megabytes. A single ZIL\_STORAGE\_READ\_ONLY object may contain a maximum of 16,000 objects (i.e., ZIL\_STORAGE\_READ\_ONLY\_OBJECT.)

ZIL\_STORAGE\_READ\_ONLY is typically used to retrieve persistent objects (e.g., objects created using OpenZinc Designer) to be retrieved. Another common use for ZIL\_STORAGE\_READ\_ONLY is as a simple data base.

The ZIL\_STORAGE\_READ\_ONLY class is declared in UI\_GEN.HPP. Its public and protected members are:

```
{struct ZIL_EXPORT_CLASS ZIL_STATS_INFO
public:
    ZIL_INT32 size;
    ZIL_INT32 createTime;
    ZIL_INT32 modifyTime;
    ZIL_UINT16 useCount;
    ZIL_UINT16 revision;
    ZIL_UINT16 countryID;
    ZIL_INODE_NUMBER inum;
};

class ZIL_EXPORT_CLASS ZIL_STORAGE_READ_ONLY : public ZIL_INTERNATIONAL
public:
    // Static file system support routines
    void StorageName(ZIL_ICHAR *buff);
    static void AppendFullPath(ZIL_ICHAR *fullPath,
        const ZIL_ICHAR *pathName = ZIL_NULLP(ZIL_ICHAR),
        const ZIL_ICHAR *fileName = ZIL_NULLP(ZIL_ICHAR) ,
        const ZIL_ICHAR *extension = ZIL_NULLP(ZIL_ICHAR));
    static void ChangeExtension(ZIL_ICHAR *name,
        const ZIL_ICHAR *newExtension);
    static void StripFullPath(const ZIL_ICHAR *fullPath,
        ZIL_ICHAR *pathName = ZIL_NULLP(ZIL_ICHAR),
        ZIL_ICHAR *fileName = ZIL_NULLP(ZIL_ICHAR) ,
        ZIL_ICHAR *objectName = ZIL_NULLP(ZIL_ICHAR),
        ZIL_ICHAR *objectPathName = ZIL_NULLP(ZIL_ICHAR));
    static void TempName(ZIL_ICHAR *tempname);
```

```

    static int ValidName(const ZIL_ICHAR *name, int createStorage = FALSE);
protected:
    static void MakeFullPath(ZIL_ICHAR *tmppath);

public:
    // Read-Only support
    static int cacheSize;
    static UI_PATH *searchPath;
    int storageError;

    ZIL_STORAGE_READ_ONLY(void);
    ZIL_STORAGE_READ_ONLY(const ZIL_ICHAR *name); // Read Only
    virtual ~ZIL_STORAGE_READ_ONLY(void);
    int ChDir(const ZIL_ICHAR *newName);
    int GetCWD(ZIL_ICHAR *path, int pathLen);
    ZIL_STORAGE_DIRECTORY *OpenDir(const ZIL_ICHAR *name);
    ZIL_STATS_INFO *Stats(void);
    int Version(void);
public:
    ZIL_ICHAR *FindFirstObject(const ZIL_ICHAR *pattern);
    ZIL_ICHAR *FindNextObject(void);
    ZIL_ICHAR *FindFirstID(ZIL_UINT16 id);
    ZIL_ICHAR *FindNextID(void);

```

## General Members

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

- *ZIL\_STATS\_INFO* contains the status of the file access operations.

*size* is the size, in bytes, of the object or file.

*createTime* contains the time when the object or file was created. *createTime* uses the C language type *time\_t*.

*modifyTime* contains the time when the object or file was last modified. *modifyTime* uses the C language type *time\_t*.

*useCount* is the number of times the object is used. MS-DOS files are used only once.

*revision* is the revision number of the file or object (i.e., the number of times that a file or object has been modified.)

*countryID* denotes the ID of the country for which the object was created. A value of 0 is used to denote the current country.

*inum* is the inode number of the object. (No further documentation of this member is provided.)

- *cacheSize* indicates how much memory is to be used as a read and write cache. The default *cacheSize* is 8 Kbytes.
- *searchPath* contains the search path for the ZIL\_STORAGE\_READ\_ONLY file when it is opened.
- *storageError* is the result of the last file access. This value is set to one of the values defined by *errno*. For more information see the global variable *errno* in the compiler language reference manual.

## ZIL\_STORAGE\_READ\_ONLY::ZIL\_STORAGE\_READ\_ONLY

### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_STORAGE_READ_ONLY(void);
```

*or*

```
ZIL_STORAGE_READ_ONLY(const ZIL_ICHAR *name);
```

### 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 both create a new ZIL\_STORAGE\_READ\_ONLY class object.

The first overloaded constructor creates a ZIL\_STORAGE\_READ\_ONLY with no associated disk file.

The second overloaded constructor creates a ZIL\_STORAGE\_READ\_ONLY and binds it to a disk file.

- *name<sub>in</sub>* is the name of the file to be opened. It may contain a path name. If the storage file cannot be opened, this function will use UI\_PATH functions to search

other paths for the file. For example, "~zinc~examples~\*.dat" will return the first **.DAT** file found with that path. Notice that the `~` character is used as the directory separator. The '\*' character is considered to be a wildcard character.

### Example

```
#include <ui_win.hpp>

UI_HELP_SYSTEM::UI_HELP_SYSTEM(char *pathName,
    UI_WINDOW_MANAGER *windowManager, UI_HELP_CONTEXT helpContext) :
    defaultHelpContext(helpContext)
{
    // Open the help storage unit.
    storage = new ZIL_STORAGE_READ_ONLY(pathName);
    if (storage->storageError)
    {
        delete storage;
        storage = NULL;
    }
}

}
```

## ZIL\_STORAGE\_READ\_ONLY::~ZIL\_STORAGE\_READ\_ONLY

### Syntax

```
#include <ui_gen.hpp>

~ZIL_STORAGE_READ_ONLY(void);
```

### Portability

This function is available on the following environments:

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

### Remarks

This destructor destroys the class information associated with the `ZIL_STORAGE_READ_ONLY` object and closes any files opened by the `ZIL_STORAGE_READ_ONLY` constructor. If a file was opened with the `UIS_TEMPORARY` flag, it will be deleted when this destructor is called.

## Example

```
#include <ui_win.hpp>

UI_HELP_SYSTEM::~UI_HELP_SYSTEM(void)
{
    if (storage)
        delete storage;
    delete helpWindow;
}
```

## ZIL\_STORAGE\_READ\_ONLY::AppendFullPath

### Syntax

```
#include <ui_gen.hpp>

static void AppendFullPath(ZIL_ICHAR *fullPath,
    const ZIL_ICHAR *pathName = ZIL_NULLP(ZIL_ICHAR),
    const ZIL_ICHAR *fileName = ZIL_NULLP(ZIL_ICHAR),
    const ZIL_ICHAR *extension = 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 function combines fragments of the path, file name and extension in order to construct a complete path name.

- *fullPath<sub>out</sub>* is the complete path name that is passed back.
- *pathName<sub>in</sub>* is the name of the path.
- *fileName<sub>in</sub>* is the name of the storage file.
- *extension<sub>in</sub>* is the extension to the file name (e.g., ".dat").

## Example

```
#include <ui_win.hpp>

UI_WINDOW_OBJECT *UI_WINDOW_OBJECT::New(const char *name,
    ZIL_STORAGE_READ_ONLY *directory,
    ZIL_STORAGE_READ_ONLY_OBJECT *file)
{
    // Check for a valid directory and file,
    int tempDirectory = FALSE;
    if (name && !file)
    {
        char pathName [128], fileName [32], objectName[32];
        ZIL_STORAGE_READ_ONLY::StripFullPath(name, pathName, fileName,
            objectName);
        if (!directory)
        {
            ZIL_STORAGE_READ_ONLY::AppendFullPath(pathName, pathName, fileName);
            ZIL_STORAGE_READ_ONLY::ChangeExtension(pathName, ".dat");
            directory = new ZIL_STORAGE_READ_ONLY(pathName, UIS_READ);
            tempDirectory = TRUE;
        }
    }
    if (!file)
        return (new UIW_WINDOW(name, directory, NULL));
}

}
```

## ZIL\_STORAGE\_READ\_ONLY::ChangeExtension

### Syntax

```
#include <ui_gen.hpp>

static void ChangeExtension(ZIL_ICHAR *name, const ZIL_ICHAR *newExtension)
```

### Portability

This function is available on the following environments:

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

### Remarks

This function changes the extension (e.g., ".dat") associated with the filename.

- *name<sub>out</sub>* is the full name of the file, possibly including the path.

- *newExtension<sub>in</sub>* is the new extension that will replace the previous extension (if any).

## Example

```
#include <ui_win.hpp>

UI_WINDOW_OBJECT *UI_WINDOW_OBJECT::New(const char *name,
    ZIL_STORAGE_READ_ONLY *directory,
    ZIL_STORAGE_READ_ONLY_OBJECT *file)
{
    // Check for a valid directory and file,
    int tempDirectory = FALSE;
    if (name && !file)
    {
        char pathName [128], fileName [32], objectName [32];
        ZIL_STORAGE_READ_ONLY::StripFullPath(name, pathName, fileName,
            objectName);
        if (!directory)
        {
            ZIL_STORAGE_READ_ONLY::AppendFullPath(pathName, pathName, fileName);
            ZIL_STORAGE_READ_ONLY::ChangeExtension(pathName, ".DAT");
            directory = new ZIL_STORAGE_READ_ONLY(pathName, UIS_READ);
            tempDirectory = TRUE;
        }
    }
    if (!file)
        return (new UIW_WINDOW(name, directory, NULL));
}
```

## ZIL\_STORAGE\_READ\_ONLY::ChDir

### Syntax

```
#include <ui_gen.hpp>

int ChDir(const ZIL_ICHAR *newName)\
```

### Portability

This function is available on the following environments:

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



## Remarks

This function changes the current working directory within a ZIL\_STORAGE\_READ\_ONLY file.

- *returnValue<sub>out</sub>* is 0 on success and -1 on failure.
- *newName<sub>in</sub>* is the name of the directory that will become the new current working directory. *newName* specifies a single sub-directory and not an entire path. The character "." is used to refer to the current working directory and ".." is used to refer to the parent directory. "." refers to the root directory if the current working directory is the root directory. The separator '/' is similar to the '\ in the DOS directory system.

## Example

```
#include <ui_win.hpp>

main(int argc, char *argv[])

{
    // Create the help directory.
    ZIL_STORAGE_READ_ONLY *helpFile = new ZIL_STORAGE_READ_ONLY(fileName,
        UIS_READWRITE);
    if (helpFile->storageError)
    {
        delete helpFile;
        helpFile = new ZIL_STORAGE_READ_ONLY(fileName,    UIS_CREATE |
            UIS_READWRITE);
    }
    helpFile->MkDir("UI_HPP");
    if (newHelpDirectory)
        helpFile->Rmdir("UI_HELP_CONTEXT");
    helpFile->MkDir("UI_HELP_CONTEXT");

    // Print genhelp status.
    PrintStatus("PROCESSING %s:\n", fileName);
    ZIL_OBJECTID helpID = 0;

    // Generate the HPP directory.
    helpFile->ChDir("~UI_HPP");
    ZIL_STORAGE_READ_ONLY_OBJECT *hppElement =
        new ZIL_STORAGE_READ_ONLY_OBJECT(*helpFile,
            "HELP_CONTEXTS", ID_HELP_CONTEXT, UIS_CREATE | UIS_READWRITE);

    // Generate the help contexts.
    helpFile->ChDir("-UI_HELP_CONTEXT");
}
```

## **ZIL STORAGE READ ONLY::FindFirstID**

### **Syntax**

```
#include <ui_gen.hpp>

ZIL_ICHAR *FindFirstID(ZIL_UINT16 id);
```

### **Portability**

This function is available on the following environments:

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

### **Remarks**

This function finds the first object in the current directory of the storage object whose *objectID* matches *id*.

- *returnValue<sub>out</sub>* is a pointer to a string containing the *stringID* of the object whose *objectID* matches *id*. If no match is found, *returnValue* is NULL.
- *id<sub>m</sub>* is the objectID of the object to be located.

## **ZIL STORAGE READ ONLY::FindFirstObject**

### **Syntax**

```
#include <ui_gen.hpp>

ZIL_ICHAR *FindFirstObject(const ZIL_ICHAR *pattern);
```

### **Portability**

This function is available on the following environments:

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

## Remarks

This function finds the first object whose *stringID* matches *pattern* inside the storage object in its current directory.

- *returnValue<sub>out</sub>* is a pointer to a string containing the *stringID* of the object that matches *pattern*. If no match is found, *returnValue* is NULL.
- *pattem<sub>in</sub>* is the stringID of an object, *pattern* may contain wildcards. For example, if *pattern* were "i\*", then **FindFirstObject**( ) would return the *stringID* of the first object that has an "i" as the first letter of *stringID*. Additionally, the '?' wildcard may be used to specify a single unknown character such as "\*.BK?".

## ZIL STORAGE READ ONLY::FindNextID

### Syntax

```
#include <ui_gen.hpp>

ZIL_ICHAR *FindNextID(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 finds the next object in the current directory in the storage whose *objectID* matches the *objectID* that was used for the most recent call to **FindFirstID**( ). This function call must be preceded by a call to **FindFirstID**( ).

- *returnValue<sub>out</sub>* is a pointer to a string containing the *stringID* of the object whose *objectID* matches the *objectID* that was used for the most recent call to **FindFirstID**( ). If no match is found, *returnValue* is NULL.

## ZIL\_STORAGE\_READ\_ONLY::FindNextObject

### Syntax

```
#include <ui_gen.hpp>

ZIL_ICHAR *FindNextObject(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 finds the next object in the current directory in the storage whose *stringID* matches the *pattern* of the last call to **FindFirstObject**( ). This function call must be preceded by a call to **FindFirstObject**( ).

- *returnValue<sub>out</sub>* is a pointer to a string containing the *stringID* of the object that matches the *pattern* of the last call to **FindFirstObject**( ). If no match is found, *returnValue* is NULL.

## ZIL\_STORAGE\_READ\_ONLY::GetCWD

### Syntax

```
#include <ui_gen.hpp>

int GetCWD(ZIL_ICHAR *path, int pathLen);
```

### 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 name of the current directory.

- *returnValue<sub>out</sub>* indicates the success of the call. If *returnValue* is zero, the call was successful. Otherwise, an error occurred.
- *path<sub>out</sub>* is the path name of the current working directory. This pointer must be a buffer allocated by the programmer.
- *pathLen<sub>in</sub>* is the length of the buffer supplied in *path*.

## ZIL STORAGE READ ONLY::MakeFullPath

### Syntax

```
#include <ui_gen.hpp>
```

```
static void MakeFullPath(ZIL_ICHAR *tmppath);
```

### 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 full path name, including a drive letter, from a partial path name.

- *tmppath<sub>in/out</sub>* is the partial path. The function modifies the contents of *tmppath* to contain the full path for the partial path originally supplied.

## **ZIL STORAGE READ ONLY::OpenDir**

### **Syntax**

```
#include <ui_gen.hpp>
```

```
ZIL_STORAGE_DIRECTORY *OpenDir(const ZIL_ICHAR *name)\
```

### **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 directory pointer within the file. This function creates a new ZIL\_STORAGE\_DIRECTORY which allows multiple pointers to be maintained within the storage file.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_STORAGE\_DIRECTORY which points to the directory opened.
- *name<sub>in</sub>* is the path name of the directory to be opened.

## **ZIL STORAGE READ ONLY::Stats**

### **Syntax**

```
#include <ui_gen.hpp>
```

```
ZIL_STATS_INFO *Stats(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 statistics regarding the `ZIL_STORAGE_READ_ONLY` object.

- *returnValue<sub>out</sub>* is a pointer to a `ZIL_STATS_INFO` structure. For more information regarding `ZIL_STATS_INFO`, see the beginning of this chapter. If an error occurs, `NULL` is returned.

## ZIL\_STORAGE\_READ\_ONLY::StorageName

### Syntax

```
#include <ui_gen.hpp>
```

```
void StorageName(ZIL_ICHAR *buff);
```

## 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 name of the file associated with the `ZIL_STORAGE_READ_ONLY` object.

- *buff<sub>in/out</sub>* is a buffer used to pass back the name of the `ZIL_STORAGE_READ_ONLY` file, *buff* should be large enough to hold the largest path plus the `NULL` terminator.

## Example

```
#include <ui_gen.hpp>

char *ExampleFunction(ZIL_STORAGE_READ_ONLY *storage)
{
    static char name[129];
    storage->StorageName (&name) ;

    return (&name);
}
```

## ZIL\_STORAGE\_READ\_ONLY::StripFullPath

### Syntax

```
#include <ui_gen.hpp>

static void StripFullPath(const ZIL_ICHAR *fullPath,
    ZIL_ICHAR *pathName=ZIL_NULLP(ZIL_ICHAR),
    ZIL_ICHAR *fileName = ZIL_NULLP(ZIL_ICHAR),
    ZIL_ICHAR *objectName = ZIL_NULLP(ZIL_ICHAR),
    ZIL_ICHAR *objectPathName = 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 function takes a full path name and divides it into its path and file name components. The arguments *fullPath*, *pathName*, *fileName*, *objectName* and *objectPathName* all have the NULL argument specified so that the information will not be saved if no other argument is provided.

- *fullPath<sub>in</sub>* is the complete path name that is passed down.
- *pathName<sub>in</sub>* is the name of the path.



- *fileName<sub>in</sub>* is the name of the storage file (including the extension).
- *objectName<sub>in</sub>* is the name of the specific object.
- *objectPathName<sub>in</sub>* is the path name for the specific object.

## Example

```
#include <ui_win.hpp>

UI_WINDOW OBJECT *UI_WINDOW OBJECT::New(const char *name,
    ZIL_STORAGE_READ_ONLY *directory,
    ZIL_STORAGE_READ_ONLY_OBJECT *file)
{
    // Check for a valid directory and file,
    int tempDirectory = FALSE;
    if (name && !file)
    {
        char pathName[128], fileName[32], objectName[32], objectPathName[128];
        ZIL_STORAGE_READ_ONLY::StripFullPath(name, pathName, fileName,
            objectName, objectPathName);
        if (!directory)
            ZIL_STORAGE_READ_ONLY::AppendFullPath(pathName, pathName, fileName);
        ZIL_STORAGE_READ_ONLY::ChangeExtension(pathName, ".dat");
        directory = new ZIL_STORAGE_READ_ONLY(pathName, UIS_READ);
        tempDirectory = TRUE;
    }
    if (!file)
        return (new UIW_WINDOW(name, directory, NULL));
}
}
```

## ZIL\_STORAGE\_READ\_ONLY::TempName

### Syntax

```
#include <ui_gen.hpp>

static void TempName(ZIL_ICHAR *tempname);
```

### 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 temporary read and write file that has a unique file name.

- *tempname<sub>out</sub>* is a buffer that will contain a unique file name that can be used as a read and write file.

## ZIL STORAGE READ ONLY::ValidName

### Syntax

```
#include <ui_gen.hpp>
```

```
static int ValidName(const ZIL_ICHAR *name, int createStorage = 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 indicates whether the specified file exists on the disk.

- *returnValue<sub>out</sub>* is TRUE if the file exists or if it can be created. Otherwise, *returnValue* is FALSE.
- *name<sub>in</sub>* is the name of the file to be checked.
- *createStorage<sub>in</sub>* when TRUE, allows the name to not actually exist as long as the directory and the file name are valid. When *createStorage* is FALSE, the file must exist in the specified path and directory.

## Example

```
#include <ui_gen.hpp>
main(int argc, char *argv[])
{
    // Make sure there is a specified text file.
```

```

    if (argc != 2)
    {
        printf("Usage: genhelp <text file name>\n");
        return (1);
    }
    // Open the text file,
    char fileName[128];
    strcpy(fileName, argv[1]);
    ZIL_STORAGE_READ_ONLY::ChangeExtension(fileName, ".txt");
    FILE *textFile = fopen(fileName, "rb");
    if (!textFile)
        printf("Could not open the text file: %s.\n", fileName);
    // Open the data file.
    ZIL_STORAGE_READ_ONLY::ChangeExtension(fileName, ".dat");
    if (!ZIL_STORAGE_READ_ONLY::ValidName(fileName, TRUE))
    {
        printf("Could not create the help data file: %s.\n", fileName);
        return (0);
    }
}
}

```

## ZIL\_STORAGE\_READ\_ONLY::Version

### Syntax

```
#include <ui_gen.hpp>
```

```
int Version(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 version of ZIL\_STORAGE\_READ\_ONLY used to create the file.

- *returnValue<sub>out</sub>* is an integer containing the ZIL\_STORAGE\_READ\_ONLY version number. The version number is 421 for version 1.0 of OpenZinc Application Framework.

## Example

```
#include <ui_gen.hpp>

ExampleFunction(ZIL_STORAGE_READ_ONLY *storage)
{
    int version = storage->Version();
    printf("The version of this data file is %d.\n", version);
}
```



## CHAPTER 71 - ZIL\_TEXT\_ELEMENT

The `ZIL_TEXT_ELEMENT` structure is used by the `ZIL_DECORATION` class to provide the text mode decorations for objects. An object's decorations are those bitmaps or characters that are used to draw an image on the object. The decorations typically include a graphical image, or bitmap, for use in graphics mode and a textual image, or character string, for use in text mode. Most environments don't require these decorations since the operating system typically provides them. OpenZinc does all the drawing in DOS and Curses, however, so these environments use decorations extensively. An example of where a decoration would be used is the maximize button. In graphics mode, it typically has a small up-arrow bitmap. In text mode, though, it usually displays a left bracket, an up-arrow character, and a right-bracket; all text characters, of course. This class maintains the text images. See "Chapter 50—`ZIL_BITMAP_ELEMENT`" for information on the bitmaps used for decorations.

The `ZIL_TEXT_ELEMENT` structure is declared in `UI_GEN.HPP`. Its public and protected members are:

```
{struct ZIL_EXPORT_CLASS ZIL_TEXT_ELEMENT
    ZIL_ICHAR *text;
    ZIL_NUMBERID numberID;
    ZIL_ICHAR stringID[ZIL_STRINGID_LEN];
};
```

### General Members

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

- *text* is the character string maintained by the `ZIL_TEXT_ELEMENT`.
- *numberID* is a numeric value used to identify the `ZIL_TEXT_ELEMENT`.
- *stringID* is a string value used to identify the `ZIL_TEXT_ELEMENT`.



## CHAPTER 72 - ZIL TIME

The ZIL\_TIME class is a lower-level class used to store and manipulate time values. It is not a window object. See "Chapter 29—UIW\_TIME" of *Programmer's Reference Volume 2* for information about the time window object.

The ZIL\_TIME class is declared on **UI\_GEN.HPP**. Its public and protected members are:

```
class ZIL_EXPORT_CLASS ZIL_TIME : public ZIL_UTIME
{
public:
    ZIL_TIME(void);
    ZIL_TIME(const ZIL_TIME &time);
    ZIL_TIME(int hour, int minute, int second = 0, int hundredth = 0);
    ZIL_TIME(const ZIL_ICHAR *string, TMF_FLAGS tmFlags = TMF_NO_FLAGS);
    ZIL_TIME(int packedTime);
    void Export(int *hour, int *minute, int *second = ZIL_NULLP(int),
               int *hundredth = ZIL_IMULLP(int));
    void Export(ZIL_ICHAR *string, TMF_FLAGS tmFlags);
    void Export(int *packedTime);
    TMI_RESULT Import(void);
    TMI_RESULT Import(const ZIL_TIME &time);
    TMI_RESULT Import(int hour, int minute, int second = 0,
                     int hundredth = 0);
    TMI_RESULT Import(const ZIL_ICHAR *string, TMF_FLAGS tmFlags);
    TMI_RESULT Import(int packedTime);

    long operator=(long hundredths);
    long operator^(const ZIL_TIME &time);
    long operator+(long hundredths);
    long operator+(const ZIL_TIME &time);
    long operator-(long hundredths);
    long operator-(const ZIL_TIME &time);
    long operator++(void);
    long operator--(void);
    void operator+=(long hundredths);
    void operator-=(long hundredths);
    int operator==(ZIL_TIME& time);
    int operator!=(ZIL_TIME& time);
    int operator>(ZIL_TIME& time);
    int operator>=(ZIL_TIME& time);
    int operator<(ZIL_TIME& time);
    int operator<=(ZIL_TIME& time);
};
```

### General Members

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



## ZIL\_TIME::ZIL\_TIME

### Syntax

```
#include <ui_gen.hpp>

ZIL_TIME(void);
    or
ZIL_TIME(const ZIL_TIME &time);
    or
ZIL_TIME(int hour, int minute, int second = 0, int hundredth = 0);
    or
ZIL_TIME(const ZIL_ICHAR *string, TMF_FLAGS tmFlags = TMF_NO_FLAGS);
    or
ZIL_TIME(int packedTime);
```

### 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 ZIL\_TIME class object.

The first overloaded constructor takes no arguments. It sets the time information according to the system's time.

The second overloaded constructor is a copy constructor that takes a previously constructed ZIL\_TIME object to specify the default time.

- *time<sub>in</sub>* is a reference pointer to a previously constructed ZIL\_TIME object.

The third overloaded constructor uses integer arguments to specify the default time.

- *hour<sub>in</sub>* is the hour. This argument must be in the range from 0 to 23.
- *minute<sub>in</sub>* is the minute. This argument must be in the range from 0 to 59.

- *second<sub>in</sub>* is the second. This argument must be in the range from 0 to 59.
- *hundredth<sub>in</sub>* is the hundredths of a second. This argument must be in the range from 0 to 99.

The fourth overloaded constructor uses a string argument to specify the default time.

- *string<sub>in</sub>* is a string that contains the time information.
- *tmFlags<sub>in</sub>* specifies how to interpret the time string. The following flags (declared in **UI\_GEN.HPP**) override the country dependent information (supplied by the operating system):

**TMF\_COLON\_SEPARATOR**—Formats the time with colons separating the time fields. For example, 12 p.m. is formatted as 12:00pm.

**TMF\_HUNDRETHS**—Formats the time with a hundredths of seconds value. For example, if the time is "12:15:10.09pm" and the TMF\_HUNDRETHS flag is set, the value "12" is interpreted as hours, the value "15" is interpreted as minutes, "10" is interpreted as seconds, and the "09" is interpreted as hundredths of seconds.

**TMF\_LOWER\_CASE**—Converts the time to lower-case.

**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.

**TMF\_NO\_HOURS**—Formats the time with no hour. For example, if the time is "12:15" and the TMF\_NO\_HOURS flag is set, the value "12" is interpreted as minutes and "15" is interpreted as seconds.

**TMF\_NO\_MINUTES**—Formats the time with no minute value. For example, if the time is "12:15" and the TMF\_NO\_MINUTES flag is set, the value "12" is interpreted as seconds and the value "15" is interpreted as hundredths of seconds.

**TMF\_NO\_SEPARATOR**—Does not place a separator between time fields.

**TMF\_SECONDS**—Formats the time with a seconds value. For example, if the time is "12:15:10" and the TMF\_SECONDS flag is set, the value "12" is

interpreted as hours, the value "15" is interpreted as minutes and "10" is interpreted as seconds.

**TMF\_SYSTEM**—Sets the time value according to the system time if the string is blank or NULL. For example, if the TMF\_SYSTEM flag is set and a NULL string value is specified, the time will be set to the system time.

**TMF\_TWELVE\_HOUR**—Formats the time using a 12 hour format, regardless of the default country information.

**TMF\_TWENTY\_FOUR\_HOUR**—Formats the time using a 24 hour format, regardless of the default country information.

**TMF\_UPPER\_CASE**—Converts the time to upper-case.

**TMF\_ZERO\_FILL**—Forces the hour, minute, second and hundredths fields to be zero filled when their values are less than 10.

The `fifth` overloaded constructor uses a packed integer argument to specify the default time.

- *packedTime<sub>n</sub>* is a packed representation of the time (whose format is the same as the MS-DOS file times). This argument is packed according to the following bit pattern:

bits 0-4 specify the seconds divided by 2 (e.g., a value of 5 means 10 seconds),  
bits 5-10 specify the minutes (0 through 59) and  
bits 11-15 specify the hours (0 through 59).

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_TIME time1;                // System time initialization.
    ZII_TIME time2(12, 0, 0);      // Integer initialization.
    ZIL_TIME *time3 = new ZIL_TIME("12:00:00pm"); // String initialization.
}
```

## ZIL\_TIME::Export

### Syntax

```
#include <ui_gen.hpp>

void Export(int *hour, int *minute, int *second = ZIL_NULLP(int),
            int *hundredth = ZIL_NULLP(int));
    or
void Export(ZIL_ICHAR *string, TMF_FLAGS tmFlags);
    or
void Export(int *packedTime);
```

### Portability

This function is available on the following environments:

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

### Remarks

The first overloaded function returns time information through the four integer arguments.

- *hour<sub>out</sub>* is a pointer to the variable that is to contain the hour. If this argument is NULL, no hour information is returned. Otherwise, this argument will be set within the range from 0 to 23.
- *minute<sub>out</sub>* is a pointer to the variable that is to contain the minute. If this argument is NULL, no minute information is returned. Otherwise, this argument will be set within the range from 0 to 59.
- *second<sub>out</sub>* is a pointer to the variable that is to contain the second. If this argument is NULL, no second information is returned. Otherwise, this argument will be set within the range from 0 to 59.
- *hundredth<sub>out</sub>* is a pointer to the variable that is to contain the hundredths of a second. If this argument is NULL, no hundredths information is returned. Otherwise, this argument will be set within the range from 0 to 99.

The second overloaded function returns the time information through the *string* argument.

*string<sub>out</sub>* is a pointer to a string that gets the formatted time. This string must be long enough to contain the time.

*tmFlags<sub>m</sub>* specifies how the return time should be formatted. The following flags (declared in UI\_GEN.HPP) override the country dependent information (supplied by the operating system):

<b>TMF_COLON_SEPARATOR</b> —Formats the time with colons separating the time fields.	12 : 00 13:00:00 12:00 a.m.
<b>TMF_HUNDREDTHS</b> —Formats the time with a hundredths of seconds value.	1:05:00.00 23:15:05.99 7:45:59.00 a.m.
<b>TMF_LOWER_CASE</b> —Converts the time to lower-case.	12:00 p.m. 1:00 a.m.
<b>TMF_NO_FLAGS</b> —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.	∞∞ 00:00 00 a.m.
<b>TMF_NO_HOURS</b> —Formats the time with no hour.	37 : 59 56:43.99
<b>TMF_NO_MINUTES</b> —Formats the time with no minute value.	12 : 56 11:45.99
<b>TMF_NO_SEPARATOR</b> —Does not place a separator between time fields.	1200 130000
<b>TMF_SECONDS</b> —Formats the time with a seconds value.	12 : 00:05 a.m. 1:13:25 <b>16:00:00</b>
<b>TMF_TWELVE_HOUR</b> —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.
<b>TMF TWENTY FOUR HOUR</b> —Formats the	

time using a 24 hour format, regardless of the default country information.

**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.

The third overloaded function returns time information through a packed integer argument.

- *packedTime<sub>out</sub>* is a packed representation of the time (whose format is the same as the MS-DOS file times). This argument is packed according to the following bit pattern:

bits 0-4 specify the seconds divided by 2 (e.g., a value of 5 means 10 seconds),  
bits 5-10 specify the minutes (0 through 59) and  
bits 11-15 specify the hours (0 through 59).

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_TIME time;        // Use a system time.

    int hour, minute, second;
    time.Export(&hour, &minute, &second);
    printf("Integer time value: hour-%d, minute-%d, second-%d\n",
           hour, minute, second);

    char stringTime[128];
    time.Export(stringTime, TMF_NO_FLAGS);
    printf("String time value: %s", stringTime);

    // The destructor for time is automatically called when the
    // scope of this function ends.
}
```

## ZIL\_TIME::Import

### Syntax

```
#include <ui_gen.hpp>

TMI_RESULT Import(void);
```

```

    or
TMI_RESULT Import(const ZIL_TIME &time);
    or
TMI_RESULT Import(int hour, int minute, int second = 0, int hundredth = 0);
    or
TMI_RESULT Import(const ZIL_ICHAR *string, TMI_FLAGS tmFlags);
    or
TMI_RESULT Import(int packedTime);

```

## 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 set the value of the ZIL\_TIME object.

The first overloaded function sets the time information according to the system time.

- *returnValue<sub>out</sub>* is the result of the import operation. *returnValue* can have one of the following values:

**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.

The second overloaded function copies the time information from the *time* reference argument.

- *returnValue<sub>out</sub>* is the result of the import operation. See the first function for possible values.
- *time<sub>in</sub>* is a reference pointer to a previously constructed time.

The third overloaded function sets the time information according to specified integer arguments.

- *returnValue<sub>out</sub>* is the result of the import operation. See the first function for possible values.
- *hour<sub>in</sub>* is the hour. This argument must be in the range from 0 to 23.
- *minute<sub>in</sub>* is the minute. This argument must be in the range from 0 to 59.
- *second<sub>in</sub>* is the second. This argument must be in the range from 0 to 59.
- *hundredth<sub>in</sub>* is the hundredths of a second. This argument must be in the range from 0 to 99.

The fourth overloaded function sets the time using information passed in a string.

- *returnValue<sub>out</sub>* is the result of the import operation. See the first function for possible values.
- *string<sub>in</sub>* is a pointer to the time string. If this is an empty string (i.e., ""), the ZIL\_TIME will be set to "blank." Passing a blank ZIL\_TIME to the **UIW\_TIME::DataSet()** function will cause the time field to be displayed as blank space. See the DataSet section of "Chapter 29—UIW\_TIME" of *Programmer's Reference Volume 2* for more information.
- *tmFlags<sub>m</sub>* specifies how the time should be formatted. The following flags (declared in **UI\_GEN.HPP**) override the country dependent information (supplied by the operating system):

**TMF\_HUNDREDTHS**—Formats the time with a hundredths of seconds value. For example, if the time is "12:15:10.09pm" and the TMF\_HUNDREDTHS flag is set, the value "12" is interpreted as hours, the value "15" is interpreted as minutes, "10" is interpreted as seconds, and the "09" is interpreted as hundredths of seconds.



**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 flag should not be used in conjunction with any other TMF flags.

**TMF\_NO\_HOURS**—Formats the time with no hour. For example, if the time is "12:15" and the TMF\_NO\_HOURS flag is set, the value "12" is interpreted as minutes and "15" is interpreted as seconds.

**TMF\_NO\_MINUTES**—Formats the time with no minute value. For example, if the time is "12:15" and the TMF\_NO\_MINUTES flag is set, the value "12" is interpreted as seconds and the value "15" is interpreted as hundredths of seconds.

**TMF\_SECONDS**—Formats the time with a seconds value. For example, if the time is "12:15:10" and the TMF\_SECONDS flag is set, the value "12" is interpreted as hours, the value "15" is interpreted as minutes and "10" is interpreted as seconds.

The  `fifth`  overloaded function sets the time information through a packed integer argument.

- `returnValueout` is the result of the import operation. See the first function for possible values.
- `packedTimein/out` is a packed representation of the time (whose format is the same as the MS-DOS file times). This argument is packed according to the following bit pattern:

bits 0-4 specify the seconds divided by 2 (e.g., a value of 5 means 10 seconds),  
bits 5-10 specify the minutes (0 through 59) and  
bits 11-15 specify the hours (0 through 59).

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_TIME time;          // Initialize a system time.

    // Import the time in various forms, then print out
    // the results.
    char stringTime[128];
    time.Import(20, 0);
    time.Export(stringTime, TMF_NO_FLAGS);
}
```

```

printf("String time value: %s\n", stringTime)

time.Import("8:00 p.m.", TMF_SECONDS);
time.Export(stringTime, TMF_TWENTY_FOUR_HOUR);
printf("String time value: %s\n", stringTime);

// The destructor for time is automatically called when the
// scope of this function ends.
}

```

## ZIL\_TIME::operator =

### Syntax

```

#include <ui_gen.hpp>

long operator = (long hundredths);
or
long operator = (const 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

The first operator overload assigns the value specified by *hundredths* to the ZIL\_TIME object.

- *returnValue<sub>out</sub>* is the number of hundredths of seconds in the resulting time. This raw value is returned so that the operator may be used in a statement containing other operations.
- *hundredths<sub>in</sub>* is the time, given in the number of hundredths of seconds, to be assigned to the ZIL\_TIME object.

The second operator overload assigns the value specified by *time* to the ZIL\_TIME object.

- *returnValue*<sub>out</sub> is the number of hundredths of seconds in the resulting time. This raw value is returned so that the operator may be used in a statement containing other operations.
- *time*<sub>in</sub> is the time to be assigned to the ZIL\_TIME object.

## Example

```
#include <ui_gen.hpp>

AddOneHour(ZIL_TIME currentTime, ZIL_ITIME &nextHour, ZIL_TIME &hourAfterNext)
{
    long oneHour = 360000L;
    ZIL_ITIME twoHours(2, 0);

    // Adding 1 hour to the current time gives the next hour.
    nextHour = currentTime + oneHour;

    // Adding 2 hour to the current time gives the following hour.
    hourAfterNext = currentTime + twoHours;
}
```

## ZIL\_TIME::operator +

### Syntax

```
#include <ui_gen.hpp>
```

```
long operator + (long hundredths);
    or
```

```
long operator + (const 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

The first operator overload adds the value *hundredths* to the ZIL\_TIME object.

- *returnValue<sub>out</sub>* is the number of hundredths of seconds resulting from the addition operation. This raw value is returned so that the operator may be used in a statement containing other operations.
- *hundredths<sub>in</sub>* is the number of hundredths of seconds to be added to the ZIL\_TIME object.

The second operator overload adds the value of *time* to the ZIL\_TIME object.

- *returnValue<sub>out</sub>* is the value resulting from the addition operation. This raw value is returned so that the operator may be used in a statement containing other operations.
- *time<sub>in</sub>* is the time to be added to the ZIL\_TIME object.

## Example

```
#include <ui_gen.hpp>

AddOneHour(ZIL_TIME currentTime, ZIL_TIME &nextHour, ZIL_TIME &hourAfterNext)
{
    long oneHour = 360000L;
    ZIL_TIME twoHours(2, 0);

    // Adding 1 hour to the current time gives the next hour.
    nextHour = currentTime + oneHour;

    // Adding 2 hours to the current time gives the following hour.
    hourAfterNext = currentTime + twoHours;
}
```

## ZIL\_TIME::operator -

### Syntax

```
#include <ui_gen.hpp>

long operator - (long hundredths);
    or
long operator - (const 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

The first operator overload subtracts the value *hundredths* from the ZIL\_TIME object.

- *returnValue<sub>out</sub>* is the number of hundredths of seconds resulting from the subtraction operation. This raw value is returned so that the operator may be used in a statement containing other operations.
- *hundredths<sub>in</sub>* is the number of hundredths of seconds to be subtracted from the ZIL\_ITIME object.

The second operator overload subtracts the value of *time* from the ZIL\_ITIME object.

- *returnValue<sub>out</sub>* is the value resulting from the subtraction operation. This raw value is returned so that the operator may be used in a statement containing other operations.
- *time<sub>in</sub>* is the time to be subtracted from the ZIL\_ITIME object.

## Example

```
#include <ui_gen.hpp>

SubtractOneHour(ZIL_TIME currentTime, ZIL_TIME &previousHour,
               ZIL_TIME &twoHoursBefore)
{
    long oneHour = 360000L;
    ZIL_TIME twoHours(2, 0);

    // Subtracting 1 hour from the current time gives the previous hour.
    previousHour = currentTime - oneHour;

    // Subtracting 2 hours from the current time gives the hour
    // two hours previous.
    twoHoursBefore = currentTime - twoHours;
```

## ZIL\_TIME::operator >

### Syntax

```
#include <ui_gen.hpp>

int operator > (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 operator overload determines whether the ZIL\_TIME object is chronologically greater than the time specified by *time*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_TIME object is chronologically greater than *time*. Otherwise, *returnValue* is FALSE.
- *time<sub>in</sub>* is the time to be compared.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_TIME currentTime; // Initialize a system time.
    ZIL_TIME midnight("12:00am");

    // Check the time,
    if (currentTime == midnight)
        printf("It's exactly midnight.\n");
    else if (currentTime > midnight)
        printf("We're in the wee hours of the morning.\n");
    else
        printf("It's still late night.\n");
}
```

## ZIL\_TIME::operator >=

### Syntax

```
#include <ui_gen.hpp>

int operator >= (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 operator overload determines whether the ZIL\_TIME object is chronologically greater than or equal to the time specified by *time*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_TIME object is chronologically greater than or equal to *time*. Otherwise, *returnValue* is FALSE.
- *time<sub>in</sub>* is the time to be compared.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_TIME currentTime; // Initialize a system time.
    ZIL_TIME midnight("12:00am");

    // Check the time,
    if (currentTime >= midnight)
        printf("Tomorrow is here.\n");
    else
        printf("It's still late night.\n");
}
```

## ZIL\_TIME::operator <

### Syntax

```
#include <ui_gen.hpp>

int operator < (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 operator overload determines whether the ZIL\_TIME object is chronologically less than the time specified by *time*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_TIME object is chronologically less than *time*. Otherwise, *returnValue* is FALSE.
- *time<sub>in</sub>* is the time to be compared.

### Example

```
#include <ui_gen.hpp>
ExampleFunction()
{
    ZIL_TIME currentTime; // Initialize a system time.
    ZIL_TIME midnight("12:00am");

    // Check the time,
    if (currentTime < midnight)
        printf("It's still late night.\n");
    else
        printf("Tomorrow is here.\n");
}
```



## ZIL\_TIME::operator <=

### Syntax

```
#include <ui_gen.hpp>

int operator <= (ZIL_ITIME &time);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload determines whether the ZIL\_ITIME object is chronologically less than or equal to the time specified by *time*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_ITIME object is chronologically less than or equal to *time*. Otherwise, *returnValue* is FALSE.
- *time<sub>in</sub>* is the time to be compared.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_TIME currentTime; // Initialize a system time.
    ZIL_TIME midnight("12:00am");

    // Check the time,
    if (midnight <= currentTime)
        printf("It's still late night.\n");
    else
        printf("Tomorrow is here.\n");
}
```

## ZIL\_TIME::operator ++

### Syntax

```
#include <ui_gen.hpp>

long operator ++ (void);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload increments the ZIL\_TIME by one hundredth of a second.

- *returnValue<sub>out</sub>* is the number of hundredths of seconds after the ZIL\_TIME object has been incremented. This raw value is returned so that the operator may be used in a statement containing other operations.

### Example

```
#include <ui_gen.hpp>
AdvanceCurrentTime(ZIL_TIME &currentTime)
{
    // Advance the current time.
    ++currentTime;
}
```

## ZIL\_TIME::operator --

### Syntax

```
#include <ui_gen.hpp>

long operator -- (void);
```

## Portability

This function is available on the following environments:

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

## Remarks

This operator overload decrements the `ZIL_TIME` by one hundredth of a second.

- *returnValue<sub>out</sub>* is the number of hundredths of seconds after the `ZIL_TIME` object has been decremented. This raw value is returned so that the operator may be used in a statement containing other operations.

## Example

```
#include <ui_gen.hpp>

DecrementCurrentTime(ZIL_TIME &currentTime)
{
    // Advance the current Time.
    --currentTime;
}
```

## ZIL\_TIME::operator +=

### Syntax

```
#include <ui_gen.hpp>

void operator += (long hundredths);
```

## Portability

This function is available on the following environments:

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

## Remarks

This operator adds *hundredths* to the ZIL\_TIME object and copies the result back into the ZIL\_TIME object.

- *hundredths<sub>in</sub>* is the number of hundredths of seconds to be added to the ZIL\_TIME object.

## Example

```
#include <ui_gen.hpp>

AddOneHour(ZIL_TIME *currentTime)
{
    long oneHour = 360000L;

    // Add 1 hour.
    #currentTime += oneHour;
}
```

## ZIL\_TIME::operator -=

### Syntax

```
#include <ui_gen.hpp>

void operator -= (long hundredths);
```

### Portability

This function is available on the following environments:

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

## Remarks

This operator subtracts *hundredths* from the ZIL\_TIME object and copies the result back into the ZIL\_TIME object.

- *hundredths<sub>in</sub>* is the number of hundredths of seconds to be subtracted from the ZIL\_TIME object.

## Example

```
#include <ui_gen.hpp>

SubtractHour(ZIL_TIME *currentTime)
{
    long oneHour = 360000L;
    // Subtract 1 hour.
    *currentTime -= oneHour;
}
```

## ZIL\_TIME::operator ==

### Syntax

```
#include <ui_gen.hpp>

int operator ==(ZIL_ITIME &time);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload determines whether the ZIL\_ITIME object is chronologically equal to the time specified by *time*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_ITIME object is chronologically equal to *time*. Otherwise, *returnValue* is FALSE.
- *time<sub>in</sub>* is the time to be compared.

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_TIME currentTime; // Initialize a system time.
    ZIL_ITIME midnight("12:00am");

    // Check the time.
}
```

```

if (currentTime == midnight)
    printf("It's exactly midnight.\n");
else if (currentTime > midnight)
    printf("We're in the wee hours of the morning.\n");
else
    printf("It's still late night.\n");

```

## ZIL\_TIME::operator !=

### Syntax

```
#include <ui_gen.hpp>
```

```
int operator != (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 operator overload determines whether the ZIL\_TIME object is chronologically not equal to the time specified by *time*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_TIME object is chronologically not equal to *time*. Otherwise, *returnValue* is FALSE.
- *time<sub>in</sub>* is the time to be compared.

### Example

```

#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_TIME currentTime; // Initialize a system time.
    ZIL_TIME startTime("12:00am");

    // Check the time.
    if ((currentTime != startTime) && (currentTime < startTime))
        printf("It's still not time yet!\n");
}

```



## CHAPTER 73 - ZIL\_UTIME

The ZIL\_UTIME class object is used to maintain and manipulate a unified time and date value.

The ZIL\_UTIME class is declared in UI\_GEN.HPP. Its public and protected members are:

```
class ZIL_EXPORT_CLASS ZIL_UTIME : public ZIL_INTERNATIONAL
{
public:
    static ZIL_ICHAR _dayName[];
    static ZIL_ICHAR _monthName[];
    static ZIL_ICHAR _timeName[];
    static int defaultInialized;

    enum UTMI_RESULT
    {
        UTMI_OK = 0,           // Time successfully imported.
        UTMI_INVALID,        // Invalid time value or format.
        UTMI_VALUE_MISSING,   // Required value not present.
        UTMI_OUT_OF_RANGE,    // Time out of range (used by UIW TIME)
        UTMI_LESS_THAN_RANGE, // Time less than positively open-ended
                             // range.
        UTMI_GREATER_THAN_RANGE // Time greater than negatively open-ended
                             // range.
    };

    ZIL_UTIME(void);
    ZIL_UTIME(const ZIL_UTIME &utime);
    ZIL_UTIME(const ZIL_ICHAR *string);
    ZIL_UTIME(int year, int month, int day, int hour, int minute,
              int second, int millisecond);
    virtual ~ZIL_UTIME(void);
    int DaysInMonth(void);
    int DaysInYear(void);
    void Export(int *year, int *month, int *day, int *hour, int *minute,
               int *second, int *milliSecond);
    int Export(ZIL_ICHAR *string, int maxsize, const ZIL_ICHAR *format);
    int Export(ZIL_ICHAR *string, int maxsize);
    UTMI_RESULT Import(void);
    UTMI_RESULT Import(const ZIL_UTIME &utime);
    const ZIL_ICHAR *Import(const ZIL_ICHAR *string,
                           const ZIL_ICHAR *format);
    int LeapYear(void);

    ZIL_UTIME *operator=(const ZIL_UTIME &utime);
    ZIL_UTIME *operator+(const ZIL_UTIME &utime);
    ZIL_UTIME *operator-(const ZIL_UTIME &utime);
    int operator==(const ZIL_UTIME &utime);
    int operator!=(const ZIL_UTIME &utime);
    int operator>(const ZIL_UTIME &utime);
    int operator>=(const ZIL_UTIME &utime);
    int operator<(const ZIL_UTIME &utime);
    int operator<=(const ZIL_UTIME &utime);

    void SetLanguage(const ZIL_ICHAR *languageName = ZIL_NULLP(ZIL_ICHAR));
    void SetLocale(const ZIL_ICHAR *localeName);

protected:
    ZIL_UINT32 jday;
    ZIL_INT32 usec;
};
```



```

int recurse;
const ZIL_LOCALE *myLocale;
const ZIL_LANGUAGE *myDayStrings;
const ZIL_LANGUAGE *myMonthStrings;
const ZIL_LANGUAGE *myTimeStrings;

void ConvertJday(int *pYear, int *pMonth, int *pDay, int *pDayOfWeek);
void ConvertUsec(int *hour, int *minute, int *second, int *milliSecond);
int DayOfWeek(void);
void Import(int year, int month, int day,
            int hour, int minute, int second, int millisecond);
void MakeCanonical(void);
public:
int basisYear;
int zoneOffset;

```

## General Members

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

- *\_dayName* is a string used to identify the ZIL\_MESSAGE\_LIST structure maintained by the ZIL\_INTERNATIONAL class that contains the strings used for day names. By default, *\_dayName* is "ZIL\_IDAY."
- *\_monthName* is a string used to identify the ZIL\_MESSAGE\_LIST structure maintained by the ZIL\_INTERNATIONAL class that contains the strings used for month names. By default, *\_monthName* is "ZIL\_MONTH."
- *\_timeName* is a string used to identify the ZIL\_MESSAGE\_LIST structure maintained by the ZIL\_INTERNATIONAL class that contains the strings used for time names. By default, *JimeName* is "ZIL\_ITME."
- *defaultInitalized* 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 *defaultInitalized* is TRUE, the strings have been set up. Otherwise they have not been, *defaultInitalized* is set to TRUE when the strings are set up in the object's constructor.
- *jday* is the Julian date representation of the date being maintained by the ZIL\_UTIME class.
- *usec* is the number of milliseconds in the time being maintained by the ZIL\_IJTIME class.
- *recurse* is a flag used when importing new date and time values from a string. *recurse* is set by the function and should not be used by the programmer.

- *myLocale* is the ZIL\_LOCALE class that contains the formatting information for dates and times.
- *myDayStrings* is the ZIL\_LANGUAGE object that contains the string translations for the days of the week.
- *myMonthStrings* is the ZIL\_LANGUAGE object that contains the string translations for the days of the month.
- *myTimeStrings* is the ZIL\_LANGUAGE object that contains the string translations for times.
- *basisYear* is the year from which dates are offset if the date does not contain a full year. For example, if *basisYear* is 1900, then a year of 90 is assumed to be 1990.
- *zoneOffset* indicates how many timezones are between the current locale and the Greenwich timezone.

## ZIL\_UTIME::ZIL\_UTIME

### Syntax

```
#include <ui_win.hpp>

ZIL_UTIME(void);
    or
ZIL_UTIME(const ZIL_IJTIME &utime);
    or
ZIL_UTIME(const ZIL_ICHAR "string");
    or
ZIL_UTIME(int year, int month, int day, int hour, int minute, int second,
    int milliSecond!);
```

### 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 ZIL\_IJTIME class object.

The first overloaded constructor creates a ZIL\_IJTIME object using the system's data and time.

The second overloaded constructor is a copy constructor that takes a previously constructed ZIL\_IJTIME object to specify the date and time.

- *utime<sub>in</sub>* is a pointer to a previously constructed ZIL\_UTIME object.

The third overloaded constructor uses a string argument to specify the date and time.

- *string<sub>in</sub>* is a string that contains the date and time information. The format must be consistent with the format specified in ZIL\_LOCALE::dateTimeStringFormat. By default, the format is "YYYY-mm-dd HH:MM:SS.KK" where Y is a year digit, m is a month digit, d is a day digit, H is an hour digit, M is a minute digit, S is a second digit and K is a thousandths of second digit.

The fourth overloaded constructor uses integer arguments to specify the date and time.

- *year<sub>in</sub>* is the year. This argument must be either 0, if no year value is to be used with the date, or a value in the range from 100 to 32,767.
- *month<sub>in</sub>* is the month. This argument must be either 0, if no month value is to be used with the date, or a value in a range from 1 (January) to 12 (December).
- *day<sub>in</sub>* is the day. This argument must be either 0, if no day value is to be used with the date, or a value in a range from 1 to 31 that should be valid for the specified month and year.
- *hour<sub>in</sub>* is the hour. This argument must be in the range from 0 to 23.
- *minute<sub>in</sub>* is the minute. This argument must be in the range from 0 to 59.
- *second<sub>in</sub>* is the second. This argument must be in the range from 0 to 59.
- *milliSecond<sub>in</sub>* is the thousandths of a second. This argument must be in the range from 0 to 999.

## ZIL\_UTIME::~~ZIL\_UTIME

### Syntax

```
#include <ui_gen.hpp>

virtual ~ZIL_UTIME(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 ZIL\_UTIME object.

## ZIL\_UTIME: .ConvertJday

### Syntax

```
#include <ui_gen.hpp>

void ConvertJday(int *pYear, int *pMonth, int *pDay, int *pDayOfWeek);
```

### Portability

This function is available on the following environments:

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

### Remarks

This function converts the Julian date into integer values for the year, month, day and day of week.

- *pYear<sub>out</sub>* is the year associated with the date.
- *pMonth<sub>out</sub>* is the month associated with the date, where 1 is January and 12 is December.
- *pDay<sub>out</sub>* is the day of the month associated with the date.
- *pDayOfWeek<sub>out</sub>* is the day of week associated with the date, where 1 is Sunday and 7 is Saturday.

## **ZIL\_UTIME::ConvertUsec**

### **Syntax**

```
#include <ui_gen.hpp>
```

```
void ConvertUsec(int *hour, int *minute, int *second, int *millisecond);
```

### **Portability**

This function is available on the following environments:

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

### **Remarks**

This function converts the time into integer values for the hour, minute, second and millisecond.

- *hour<sub>out</sub>* is the hour associated with the time.
- *minute<sub>out</sub>* is the minute associated with the time.
- *second<sub>out</sub>* is the second associated with the time.
- *millisecond<sub>out</sub>* is the millisecond associated with the time.

## ZIL\_UTIME::DayOfWeek

### Syntax

```
#include <ui_gen.hpp>

int DayOfWeek(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 numerical value of the day of the week (Sunday = 1, Monday = 2, . . . Saturday = 7) for the ZIL\_UTIME object.

**NOTE:** `DayOfWeek()` may return questionable values for dates before 1753 due to the switch from the Julian calendar to the Gregorian calendar.

- *returnValue<sub>out</sub>* is the day of the week.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_UTIME utime;

    // Print the current day of week.
    printf("Today is %d.\n", utime.DayOfWeek());
}
```

## ZIL\_UTIME::DaysInMonth

### Syntax

```
#include <ui_gen.hpp>

int DaysInMonth(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 number of days in the month specified by the ZIL\_UTIME object. For example, if the date were December 15, 1993, **DaysInMonth** would return 31.

**NOTE:** **DaysInMonth( )** may return questionable values for dates before 1753 due to the switch from the Julian calendar to the Gregorian calendar.

- *returnValue<sub>out</sub>* is the number of days in the month.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    // Print the number of days in the current month.
    ZIL_UTIME utime;
    printf("This month has %d days.Xn", utime.DaysInMonth());
}
```

## ZIL\_UTIME::DaysInYear

### Syntax

```
#include <ui_gen.hpp>

int DaysInYear(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 number of days in the year specified by the ZIL\_UTIME object. For example, if the date were January 15, 1992, DaysInYear() would return 366 (i.e., 1 extra day for leap year).

NOTE: DaysInYear( ) may return questionable values for dates before 1753 due to the switch from the Julian calendar to the Gregorian calendar.

- *returnValue<sub>out</sub>* is the number of days in the year.

### Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    // Print the number of days in the year.
    ZIL_UTIME utime;
    printf("This year has %d days.\n", utime.DaysInYear());
}
```



## ZIL\_UTIME::Export

### Syntax

```
#include <ui_gen.hpp>

void Export(int *year, int *month, int *day, int *hour, int *minute,
            int *second, int *milliSecond);
    or
int Export(ZIL_ICHAR *string, int maxSize, const ZIL_ICHAR *format);
    or

int Export(ZIL_ICHAR *string, int maxSize);
```

### 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 obtain the value of the ZIL\_UTIME object.

The first overloaded function returns date and time information through several integer arguments.

- *year<sub>out</sub>* is the year associated with the date.
- *month<sub>out</sub>* is the month associated with the date.
- *day<sub>out</sub>* is the day of the month associated with the date.
- *hour<sub>out</sub>* is the hour associated with the time.
- *minute<sub>out</sub>* is the minute associated with the time.
- *second<sub>out</sub>* is the second associated with the time.
- *milliSecond<sub>out</sub>* is the millisecond associated with the time.

The second overloaded function returns date information through the *string* argument, using *format* to format the string.

- *returnValue<sub>out</sub>* indicates if the export was successful. *returnValue* is non-zero if the export was successful. Otherwise, the export was unsuccessful.
- *string<sub>in/out</sub>* is a pointer to a string that gets the formatted utime. This string must be long enough to contain the combined time and date.
- *maxSize<sub>in</sub>* is the size of the string buffer.
- *format<sub>in</sub>* specifies how the string is to be formatted, *format* is a **printf**-style format string that uses special symbols to represent the various possible fields.

The third overloaded function returns the date information through the *string* argument.

- *returnValue<sub>out</sub>* indicates if the export was successful. *returnValue* is non-zero if the export was successful. Otherwise, the export was unsuccessful.
- *string<sub>out</sub>* is a pointer to a string that gets the formatted utime. This string must be long enough to contain the combined time and date.
- *maxSize<sub>in</sub>* is the size of the string buffer.

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_UTIME utime;

    // Print out the time and date.
    char stringUtime[128];
    utime.Export(stringUtime, 128);
    printf("String value: %s", stringUtime);

    // The destructor for utime is automatically called when the
    // scope of this function ends.
}
```

## ZIL\_UTIME::Import

### Syntax

```
#include <ui_gen.hpp>

UTMI_RESULT Import(void);
    or
UTMI_RESULT Import(const ZIL_UTIME &utime);
    or
const ZIL_ICHAR *Import(const ZIL_ICHAR *string, const ZIL_ICHAR *format);
    or
void Import(int year, int month, int day, int hour, int minute, int second, int millisecond);
```

### 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 set the value of the ZIL\_IJTIME object.

The first overloaded function sets the date and time information according to the system information.

- *returnValue<sub>out</sub>* is the result of the import operation. *returnValue* can have one of the following values:

**UTMI\_GREATER\_THAN\_RANGE**—The date or time value was greater than the maximum value in a range that goes to negative infinity.

**UTMI\_INVALID**—An invalid format was encountered (e.g., "31 Jan, 1992").

**UTMI\_LESS\_THAN\_RANGE**—The date or time value was less than the minimum value in a range that goes to positive infinity.

**UTMI\_OK**—The date and time were entered in a correct format and within the valid range.

**UTMI\_OUT\_OF\_RANGE**—The date or time value was out of range (e.g., "Jan 33, 1992").

**UTMI\_VALUE\_MISSING**—A required date or time value was missing (e.g., "5, 1991").

The second overloaded function copies the date and time information from the *utime* reference argument.

- *returnValue<sub>out</sub>* is the result of the import operation. See the first function for possible values.
- *utime<sub>in</sub>* is a pointer to a previously constructed utime.

The third overloaded function copies the time and date information from the *string* argument.

- *returnValue<sub>out</sub>* is the date and time string that resulted from the import.
- *string<sub>in</sub>* is a string that contains the date and time information.
- *format<sub>in</sub>* is a **printf**-style format string that specifies how the date and time information can be parsed from *string*.

The fourth overloaded function copies the date and time information from the specific date and time values.

- *year<sub>in</sub>* is the year to associate with the date.
- *month<sub>in</sub>* is the month to associate with the date.
- *day<sub>in</sub>* is the day to associate with the date.
- *hour<sub>in</sub>* is the hour to associate with the time.
- *minute<sub>in</sub>* is the minute to associate with the time.
- *second<sub>in</sub>* is the second to associate with the time.
- *millisecond<sub>in</sub>* is the millisecond to associate with the time.

## Example

```
#include <ui_gen.hpp>

ExampleFunction()
{
    ZIL_UTIME utime;

    // Import the time and date and print out the results,
    char stringUtime[128];
    utime.Import("1994-1-28 1:35:00.0", defaultLocale->dateTimeStringFormat);
    utime.Export(stringUtime, 128);
    printf("String value: %s\n", stringUtime);

    // The destructor for utime is automatically called when the
    // scope of this function ends.
}
```

## ZIL\_UTIME::LeapYear

### Syntax

```
#include <ui_gen.hpp>
```

```
int LeapYear(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 indicates if the year is a leap year.

- *returnValue<sub>out</sub>* indicates if the year is a leap year. *returnValue* is TRUE if the year is a leap year. Otherwise it is FALSE.

## ZIL\_UTIME::MakeCanonical

### Syntax

```
#include <ui_gen.hpp>

void MakeCanonical(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 ensures that *jday* and *usec* are synchronized. For example, if a number of milliseconds are added to the utime, *usec* may contain a value greater than the number of milliseconds in a day. If this happens, *jday* needs to be updated by an extra day and *usec* needs to be updated so that it represents a proper time for the date.

## ZIL\_UTIME::SetLanguage

### Syntax

```
#include <ui_gen.hpp>

void SetLanguage(const ZIL_ICHAR *languageName = 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 function sets the language to be used by the object. The string translations for the object will be loaded and the object's *myDayStrings*, *myMonthStrings*, and *myTimeStrings* members will be updated to point to the new ZIL\_LANGUAGE objects. 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 ZINC\SOURCE\INTL directory to the \ZINC\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<sub>in</sub>* is the two-letter ISO language code identifying which language the object should use.

## ZIL\_UTIME::SetLocale

### Syntax

```
* #include <ui_gen.hpp>

void SetLocale(const ZIL_ICHAR *localeName)\
```

### 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 locale to be used by the object. The locale information for the object will be loaded and the object's *myLocale* member will be updated to point to the new ZIL\_LOCALE object. By default, the object uses the locale identified in the LOC\_DEF.CPP file, which compiles into the library. (If a different default locale is desired, simply copy a LOC\_<ISO>.CPP file from the ZINC\SOURCE\INTL directory to the \ZINC\SOURCE directory, and rename it to LOC\_DEF.CPP before compiling the library.) The locale information is loaded from the I18N.DAT file, so it must be shipped with your application.

- *localeName<sub>cn</sub>* is the two-letter ISO country code identifying which locale information the object should use.

## **ZIL\_UTIME::operator =**

### **Syntax**

```
#include <ui_gen.hpp>

ZIL_UTIME *operator=(const ZIL_UTIME &utime);
```

### **Portability**

This function is available on the following environments:

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

### **Remarks**

This operator overload assigns the value specified by *utime* to the ZIL\_IJTIME object.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_IJTIME object after it has been updated. This pointer is returned so that the operator may be used in a statement containing other operations.
- *utime<sub>in</sub>* is the time and date to be assigned to the ZIL\_IJTIME object.

### **Example**

```
#include <ui_gen.hpp>

AssignUtime(ZIL_UTIME aUtime, ZIL_UTIME &anotherUtime)
{
    anotherUtime = aUtime;
}
```



## ZIL\_UTIME-operator +

### Syntax

```
#include <ui_gen.hpp>
```

```
ZIL_UTIME *operator +(const ZIL_UTIME &utime);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload adds the value contained in *utime* to the ZIL\_UTIME object.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_UTIME object after it has been updated. This pointer is returned so that the operator may be used in a statement containing other operations.
- *utime<sub>in</sub>* is the time and date to be added to the ZIL\_UTIME object.

### Example

```
#include <ui_gen.hpp>
AddUtime(ZIL_IJTIME aUtime, ZIL_IJTIME kanotherUtime)
{
    ZIL_UTIME currentUtime;
    anotherUtime = aUtime + currentUtime;
}
```

## ZIL\_UTIME::operator -

### Syntax

```
#include <ui_gen.hpp>

ZIL_UTIME *operator -(const ZIL_UTIME &utime);
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload subtracts the value contained in *utime* from the ZIL\_UTIME object.

- *returnValue<sub>out</sub>* is a pointer to the ZIL\_UTIME object after it has been updated. This pointer is returned so that the operator may be used in a statement containing other operations.
- *utime<sub>in</sub>* is the time and date to be subtracted from the ZIL\_UTIME object.

### Example

```
#include <ui_gen.hpp>

SubtractUtime(ZIL_UTIME aUtime, ZIL_UTIME &anotherUtime)
{
    ZIL_UTIME currentUtime;
    anotherUtime = aUtime - currentUtime;
}
```

## **ZIL\_UTIME:operator >**

### **Syntax**

```
#include <ui_gen.hpp>

int operator > (const ZIL_UTIME &utime);
```

### **Portability**

This function is available on the following environments:

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

### **Remarks**

This operator overload determines whether the ZIL\_UTIME object is chronologically greater than the utime specified by *utime*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_UTIME object is chronologically greater than *utime*. Otherwise, *returnValue* is FALSE.
- *utime<sub>in</sub>* is the time and date to be compared.

## **ZIL\_UTIME::operator >=**

### **Syntax**

```
#include <ui_gen.hpp>

int operator >= (const ZIL_UTIME &utime);
```

### **Portability**

This function is available on the following environments:

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

## Remarks

This operator overload determines whether the ZIL\_UTIME object is chronologically greater than or equal to the utime specified by *utime*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_UTIME object is chronologically greater than or equal to *utime*. Otherwise, *returnValue* is FALSE.
- *utime<sub>in</sub>* is the time and date to be compared.

## ZIL\_UTIME::operator <

### Syntax

```
#include <ui_gen.hpp>

int operator < (const ZIL_UTIME &utime);
```

### Portability

This function is available on the following environments:

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

## Remarks

This operator overload determines whether the ZIL\_IJTIME object is chronologically less than the utime specified by *utime*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_UTIME object is chronologically less than *utime*. Otherwise, *returnValue* is FALSE.
- *utime<sub>in</sub>* is the time and date to be compared.

## ZIL\_UTIME:operator <=

### Syntax

```
#include <ui_gen.hpp>
```

```
int operator <= (const ZIL_UTIME &utime)\
```

### Portability

This function is available on the following environments:

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

### Remarks

This operator overload determines whether the ZIL\_IJTIME object is chronologically less than or equal to the utime specified by *utime*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_IJTIME object is chronologically less than or equal to *utime*. Otherwise, *returnValue* is FALSE.
- *utime<sub>m</sub>* is the time and date to be compared.

## ZIL\_UTIME::operator ==

### Syntax

```
#include <ui_gen.hpp>
```

```
int operator == (const ZIL_UTIME &utime)\
```

### Portability

This function is available on the following environments:

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

## Remarks

This operator overload determines whether the ZIL\_UTIME object is chronologically equal to the utime specified by *utime*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_IJTIME object is chronologically equal to *utime*. Otherwise, *returnValue* is FALSE.
- *utime<sub>in</sub>* is the time and date to be compared.

## ZIL\_UTIME::operator !=

### Syntax

```
#include <ui_gen.hpp>

int operator != (const ZIL_UTIME &utime);
```

### Portability

This function is available on the following environments:

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

## Remarks

This operator overload determines whether the ZIL\_IJTIME object is chronologically not equal to the utime specified by *utime*.

- *returnValue<sub>out</sub>* is TRUE if the ZIL\_IJTIME object is chronologically not equal to *utime*. Otherwise, *returnValue* is FALSE.
- *utime<sub>in</sub>* is the time and date to be compared.



# INDEX

- != (operator overload)
  - bignum implementation of 522
  - date implementation of 556
  - position implementation of 298
  - region implementation of 334
  - time implementation of 737
- + (operator overload)
  - bignum implementation of 514
  - date implementation of 545
  - list implementation of 223
  - time implementation of 726
- ++ (operator overload)
  - bignum implementation of 517
  - date implementation of 551
  - position implementation of 303
  - region implementation of 335
  - time implementation of 733
- += (operator overload)
  - bignum implementation of 519
  - date implementation of 553
  - position implementation of 305, 337
  - region implementation of 337
  - time implementation of 734
- (operator overload)
  - bignum implementation of 515
  - date implementation of 546
  - list implementation of 234
  - time implementation of 727
- (operator overload)
  - bignum implementation of 518
  - date implementation of 552
  - position implementation of 304, 336
  - time implementation of 733
- = (operator overload)
  - bignum implementation of 520
  - date implementation of 554
  - position implementation of 306
  - region implementation of 338
  - time implementation of 735
- \* (operator)
  - bignum implementation of 516
- < (operator overload)
  - bignum implementation of 525
  - date implementation of 549
  - position implementation of 299
  - time implementation of 731
- <= (operator overload)
  - bignum implementation of 526
  - date implementation of 550
  - position implementation of 302
  - time implementation of 732
- = (operator overload)
  - bignum implementation of 513
  - date implementation of 544
  - time implementation of 725
- == (operator overload)
  - bignum implementation of 521
  - date implementation of 555
  - position implementation of 297
  - region implementation of 333
  - time implementation of 736
  - UI\_REGION implementation of 333
- > (operator overload)
  - bignum implementation of 523
  - date implementation of 546-548, 730
  - position implementation of 300
  - time implementation of 729
- >= (operator overload)
  - bignum implementation of 524
  - date implementation of 548
  - position implementation of 301
  - time implementation of 730
- &
  - hotkey designator 609
  - atcFlags 25-27
  - backgroundColor 196, 198, 252, 254, 376, 378
  - basisYear 529, 543
  - blankString 614
  - bottom 342
  - className 139, 180, 183, 205, 403, 404, 415, 482, 567, 592, 630, 632, 646
  - dayName 740
  - denominator 353, 354
  - dncFlags 71, 72
  - doDelete 649, 650



- `_errorPaletteMapTable` 282
- `_errorString` 614
- `_fileName` 308, 312, 314, 315
- `_fill` Attributes 196, 198
- `_fillPattern` 196, 198, 252, 254, 376, 378
- `_foregroundColor` 196, 198, 252, 254, 376, 378
- `_graphicSwitches` 16, 17
- `_helpPaletteMapTable` 282
- `_langName` 615
- `_left` 342
- `_locale` 615
- `_locName` 615
- `_manager` 45, 48, 49
- `_mapName` 649-651
- `_maximum` 71, 72
- `_minimum` 71, 72
- `_mode` 369, 370
- `_monthName` 740
- `_moveBuffer` 370
- `_normalPaletteMapTable` 281
- `_numberID` 400, 426, 427
- `_numerator` 353, 354
- `_object` 25, 26, 45, 46, 71, 72, 353, 354, 569, 575
- `_offset` 25, 26, 353, 354
- `_outlineAttributes` 196, 198
- `_path` 581, 587
- `_pathString` 283
- `_position` 671, 673, 683, 689
- `_reference` 25, 26
- `_region` 341, 342
- `_right` 342
- `_rlcFlags` 353, 354
- `_screen` 369, 370
- `_screenID` 342
- `_stopDevice` 38, 40, 196, 198, 252, 254, 369, 370, 376, 378
- `_tComerLL` 373
- `_tComerLR` 373
- `_tComerUL` 372
- `_tComerUR` 372
- `_textSwitches` 16, 17
- `_tHorizontal` 373
- `_timeName` 740
- `_top` 342

- `-tVertical` 373
- `_type` 126, 146, 148
- `_virtualCount` 38, 40, 196, 198, 252, 254, 369, 370, 376, 378
- `_virtualRegion` 38, 40, 196, 198, 252, 254, 369, 370, 376, 378
- `_zincPathString` 283

## A

- `abs` (function)
  - bignum implementation of 504
- `absolute` 400, 406, 409, 441, 442, 451, 504
- `absolute value`
  - dates 531, 742
  - time 716
- `abstract classes`
  - device implementation of 59
  - display implementation of 83
- `active object` 424
- `Add` (function) 159, 181, 223, 241, 387
  - list-block implementation of 241
- `adding windows to window manager` 387
- `allocate` 238, 406, 595, 603-606, 611, 612, 649, 653, 656, 657
- `allocateBelow` 347, 350
- `ALT.STATE` 60
- `altDigits` 639, 641
- `ANSI functions` 595
- `appClass` 85, 89, 459, 461, 462
- `appContext` 85, 88
- `AppendFullPath` (function) 697
- `apple About` 245, 248
- `appleMenu` 245, 247
- `argc` 16-19, 284, 285, 459, 461, 462, 665, 667, 700, 709, 710
- `args` 413, 446
- `argv` 16-19, 459, 461, 462
- `arrays`
  - of class objects 237
  - of event information 318, 324
  - pop-up item definition 213
- `Assign` (function) 296, 328

- position implementation of 296
- region implementation of 328
- AssignData (function) 558, 583, 618, 634
- atcFlags 25-27
- autodetection
  - graphics drivers 41, 199, 255, 379
  - text screen 371

## B

- backgroundPalette 41, 85, 88, 199, 254, 278, 282, 371, 378
- backup 86, 89, 663, 669, 670
- base class 125, 219
- baseCallback 402
- basisYear 529, 543, 740, 741
- Beep (function) 134
- BGI display 37
- BGIFONT 37-39, 42
- BGIFONT (structure) 38
- BGIPATTERN 37, 39
- Bitmap (function) 92
- Bitmap (virtual function) 92
- bitmapArray 37, 85, 92, 95, 97, 195, 245, 251, 259, 265, 271, 308, 375, 459, 460
- BitmapArrayToHandle (function) 95
- BitmapArrayToHandle (virtual function) 95
- BitmapHandleToArray (function) 97
- BitmapHandleToArray (virtual function) 97
- bitmapHeight 37, 85, 92, 95, 97, 195, 245, 251, 259, 265, 271, 308, 375, 459, 460
- bitmapWidth 37, 85, 92, 95, 97, 195, 245, 251, 259, 265, 271, 308, 375, 459, 460
- blinkRate 465, 466, 469
- bnumLeftParen 639, 641
- bnumRightParen 639, 641
- border
  - text mode 372
- Borland

## Index

- graphics display 37
- breakHandlerSet 473
- buff 569, 571, 575, 577, 675, 680, 683, 687, 690, 693, 706
- bwBackground 104, 277, 278

## C

- C library
  - overloaded functions 593
- cacheSize 694, 695
- canonicalLocale 595, 613, 614
- cascading windows 392
- ceil (function)
  - bignum implementation of 504
- cellHeight 85, 87, 117, 282, 296, 332
- cellWidth 85, 87, 296, 332
- Center (function) 388
- ChangeExtension (function) 698
- character
  - context-dependent 602
  - hotkey 609
  - size 607, 654
  - spacing 602
  - text mode windows 372
- character set
  - hardware 607, 612, 655, 657
  - mapping 604-606, 652, 653
  - multi-byte 607, 654
  - Unicode 607, 612, 655, 657
  - unmapping 611, 656
- characters
  - high intensity 369
- CharMapInitialize (function) 596
- charSize 37, 38
- chartod (function) 597
- ChDir (function) 699
- class arrays
  - list block implementation of 238
- ClassLoadData (function) 561, 585, 621, 635
- className 125, 128, 139, 179, 180, 182, 183, 205, 400-404, 415, 444, 482,

- 563, 566, 567, 581, 589, 591, 592, 627, 630-632, 646
- ClassName (function) 128, 182, 415
- classRegister 446
- classRegistered 444
- ClassStoreData (function) 562, 585, 622, 636
- clip 100, 107, 282, 350, 401, 412, 443
- clipping 341, 347
- clipRegion 37, 38, 85, 86, 92, 94, 98, 99, 102, 104-107, 109-111, 114, 116, 117, 195, 196, 245, 246, 251, 252, 259, 260, 265, 266, 271, 272, 308, 369, 375, 376, 459, 460
- codeSet 90, 595, 615
- color palettes
  - black & white 278
  - gray-scale 278
- colorAttribute 277
- colorBackground 104, 277, 278
- colorBitmap 37, 85, 92, 94-98, 195, 245, 251, 259, 265, 271, 308, 375, 459, 460
- colorForeground 104, 277, 278
- colorMap 85, 88
- colors 435
- columns 65, 66, 85, 87, 197, 343, 350, 351, 371, 424
- compare function
  - definition of 220, 239
- CompareDevices (function) 64
- compareFunction 219-221, 223, 226, 233, 238, 239, 241
- compareFunctionName 404
- Control (function) 20
- control-break
  - setting 473
- controlData 402, 444
- controlScreenID 400, 405
- convenienceFunction 402, 444, 446
- ConvertFromFilename (function) 597
- converting
  - multi-byte string 607, 612, 655, 657
  - Unicode 607, 612, 655, 657
  - wide-character string 607, 612, 655, 657
- ConvertJday (function) 743

- ConvertToFilename (function) 598
- ConvertUsec (function) 744
- coordinates
  - cursor 466
- copy constructor
  - date use of 530
- Count (function) 224
  - list implementation of 224
- countdown timer 493
- country ID 693, 694
- CreateData (function) 564, 589, 628, 644
- CreateMotifString (function) 415
- createStorage 694, 709
- createTime 693, 694
- Ctype functions 595
- currency
  - format 640
- currencySymbol 640
- Current (function) 159, 183, 225, 285, 319, 348
  - list implementation of 225, 319, 348
- cursor 465
  - appearance 465
  - position on screen 466

## D

- data file
  - path to 283
  - traversing 588
- data files
  - finding objects 701-703
  - linking 664
  - naming 669, 706
  - removing directories 662, 667
  - renaming objects 666
  - saving 669
  - saving internal buffers 663
  - saving objects 668
  - statistics 705
  - valid recognition 709
  - versions 710

- date 3, 5, 6, 9, 10, 13, 176, 279, 403, 405, 529, 530-556, 641, 739-744, 746-753, 755-761, 529
  - format 641
  - low-level 529
- dates
  - absolute value 531
  - alphanumeric 531
  - Asian format 531, 542
  - European formats 531, 542
  - format flags 531
  - international formats 531
  - military formats 531, 542
  - system 502, 530
  - U.S. formats 532, 543
- dateSeparator 641
- dateStringFormat 639, 641
- dateTimeStringFormat 639, 641, 742, 752
- dayOfWeek 529, 533, 535, 740, 745
- DayOfWeek (function) 533, 745
- days 529, 534, 535, 544-547, 552-554, 741, 746, 747
- DaysInMonth (function) 534, 746
- DaysInYear (function) 534, 747
- decimalSeparator 639
- decimalString 499, 501, 505, 509
- DecomposeCharacter (function) 599
- DecomposeString (function) 600
- decorations 481, 527, 557, 563, 565-568, 713
- decorationsName 565, 567
- defaultBitmap 563, 566
- defaultCallback 401, 412
- defaultCharMap 595, 596, 606, 607, 611, 612, 614
- defaultDateFlags 639, 641
- defaultHelpContext 205, 206, 661, 696
- Default18nInitialize (function) 600
- defaultInitialized 139, 205, 481, 739, 740
- defaultLocale 595, 601, 613, 643, 646, 752
- defaultMessages 627, 630
- defaultName 589
- defaults 459, 581, 582
- defaultStatus 399, 402
- defaultStorage 143, 400, 403, 581, 582, 588, 403
- defaultText 563, 566
- defaultTimeFlags 639, 641
- defDigits 639, 641
- defProcInstance 401, 444
- DeleteData (function) 559, 584, 619, 635
- delta 4, 6, 13, 365, 366, 569-571, 575, 576, 577
- deltaX 481,490
- deltaY 481,490
- denominator 353-355, 357, 358
- depth 118, 120, 394, 395, 402, 419, 427, 430, 431
- derived classes
  - from base window object 399
  - from device base class 59
- Destroy (function) 226
  - list implementation of 226
- DestroyObject (function) 662
- DestroyObject(function) 662
- detectgraph( ) (function) 41
- detection
  - graphics drivers 41, 199, 255, 379
- device1 60, 64
- device2 60, 64
- deviceImage 156, 160-162
- DeviceImage (function) 160
- DevicePosition (function) 162
- devices
  - hiding 65
  - turning on/redisplaying 66
- deviceState 156, 164, 166, 470, 476, 486, 489
- DeviceState (function) 164
- deviceType 155, 156, 160, 162, 164-166
- digits 499, 500, 507, 640, 641
- dirSepStr 650
- display 83
  - abstract definition 83
  - Borland BGI 37
  - Microsoft Windows 259
  - OS/2 271
  - programmer defined 84
  - text display 369
  - UI\_APPLICATION 15

- display management 83
- DisplayHelp (function) 204, 209
  - help system implementation of 209
- displayImage 90
- displayPort 307, 309
- displayText 415
- divX 37, 38
- divY 37, 38
- dncFlags 71,72,74
- doubleClickRate 399, 402
- dragObject 383, 384
- DrawBorder (function) 416
  - window object implementation of 416, 426
- DrawItem (function) 418
  - window object implementation of 418
- DrawShadow (function) 419
  - window object implementation of 419
- DrawText (function) 420
- driver 37, 40, 41, 61
- dst 595, 597, 598
- dtFlags 529, 530, 535, 540
- dwStyle 401,412,442

## E

- element 125
- element 1 127, 128, 219, 220, 227, 230, 235, 239, 240
- element2 127, 128, 220, 227, 230, 235, 239, 240
- elementArray 238-241, 318, 319, 324
- Ellipse (function) 98
- Ellipse (virtual function) 98
- Encompassed (function) 329
- endAngle 37, 85, 98, 99, 195, 245, 251, 259, 265, 271, 308, 375, 460
- environment variable
  - PATH 283
  - ZINC PATH 283
- erno 684, 695
- error management 133,139
- ErrorMessage (function) 134, 142
- errors

- initialized devices 60, 61
- errorStatus 133-137, 139, 142, 143
- errorSystem 140, 141, 143, 208, 210, 400, 403
- European date formats 531, 542
- event
  - interpretation 390, 422
- Event (function) 65, 166, 184, 389, 421, 468, 477, 487, 497
- Event (virtual function) 421
  - cursor implementation of 468
  - device implementation of 65
  - event manager implementation of 166
  - keyboard implementation of 477
  - mouse implementation of 487, 490
  - window manager implementation of 389
  - window object implementation of 421
- event flow 389
- Event Manager
  - UI\_APPLICATION 15
- event mapping 175, 434
- event processing 155
- eventMapTable (static variable) 112, 178, 400, 404
- events
  - logical mapping 175
  - mouse 483, 487
- eventType 175, 176
- exit function 384
- Export (function) 505, 535, 719, 748
  - bignum implementation of 505
  - date implementation of 535
  - time implementation of 719
- extension 669, 670, 693, 697-699, 708
- extraName 595, 604, 649, 650

## F

- face 245, 246, 307
- fallbackResources 459, 461
- file support functions 595
- FILECHAR
  - type 598
- filename

- conversion 598
- files
  - access 660
  - closing 661, 696
  - extensions 698
  - opening 660, 695
- fill pattern
  - UI\_BGI\_DISPLAY implementation of 39
  - UI\_MSC\_DISPLAY implementation of 252
  - OpenZinc implementation of 195, 197
- fillCharacter 277
- fillLine 594,609
- fillRegion 402, 416, 417, 419-421
- FindFirstID (function) 701
- FindFirstObject (function) 701
- findFunction 219, 228, 229
- FindNextID (function) 702
- FindNextObject (function) 703
- First (function) 167, 185, 227, 286, 320, 348
- FirstPathName (function) 287
- flags
  - advanced window objects 127, 282, 400, 402, 407, 413, 414, 407
  - general window objects 405
- flFlag 401,413,446
- floor (function) 507
- fl Style 401,412
- Flush (function) 663
- font 7, 37-39, 41-43, 86, 87, 116-119, 195, 196, 197, 201, 245-247, 252, 253, 256, 260, 265-267, 272, 274, 275, 283, 307-310, 369, 376, 377, 380, 400, 401, 412, 426, 459-461
- Font (function) 426
- fontList 459, 460
- fonts
  - default 41, 199, 255, 259, 271
  - UI\_BGI\_DISPLAY implementation of 38
  - UI\_GRAPHICS\_DISPLAY implementation of 195-197, 196
  - UI\_MACINTOSH\_DISPLAY implementation of 247
  - UI\_MOTIF\_DISPLAY implementation of 459,460
  - UI\_MSC\_DISPLAY implementation of 252
  - UI\_MSWINDOWS\_DISPLAY implementation of 260
  - UI\_NEXTSTEP\_DISPLAY implementation of 266
  - UI\_OS2\_DISPLAY implementation of 272
  - UI\_WCC\_DISPLAY implementation of 377
  - UI\_XT\_DISPLAY implementation of 461
- fontSet 459,461
- fontStruct 459, 460
- fontTable 39, 41, 43, 117, 197, 201, 247, 253, 256, 260, 266, 272, 275, 309, 377, 380, 412, 461
- forcelInitialization 593, 608
- foreground 44, 99, 103, 105, 106, 108, 109, 116, 198, 202, 254, 257, 265, 269, 277, 278, 378, 381, 421, 459
- fractionDigits 639, 640
- fRec 245, 247, 307
- free list 237
- FreeDecorations (function) 565
- FreeI18N (function) 590
- FreeLanguage (function) 628
- freeList 238-240, 318, 324
- FreeLocale (function) 644
- fromColor 86, 246, 249
- fromStandard 649-651
- Full (function) 242
- fullPath 693, 697, 707

**G**

- geometry management 353
- Get (function) 168, 228, 426
  - event manager implementation of 168
  - list implementation of 228
  - window object implementation of 426
- GetBasis (function) 539

GetBitmap (function) 559  
 GetCWD (function) 703  
 GetLocale (function) 508  
 GetMessage (function) 619  
 GetText (function) 560  
 graphics  
   bar 107  
   bit image 101  
   bit images 92, 95, 97, 100  
   bitmap 92, 95, 97, 100, 101  
   Borland BGI 37  
   circle 98  
   ellipse 98  
   fill patterns 196, 198, 252, 254, 277,  
     278, 376, 378, 278  
   fill region 107  
   icon 100, 101  
   line 102  
   Microsoft Windows 259  
   OS/2 271  
   palette mapping 279  
   palettes 277  
   polygon 105  
   rectangle 107  
   text 116  
   VirtualGet 120  
   VirtualPut 122  
 graphics mode 199  
 GRAPHICSFONT (structure) 195-197,  
   196  
 grayScaleBackground 104, 277, 278  
 grouping 640  
  
 hardware configuration  
   determining 614, 615  
 hdc 259, 260, 307, 309  
 Height (function) 330  
 help contexts 410  
 help file format 206  
 help management 205  
 helpContext 203-205, 207, 209, 400, 410,  
   661, 696  
 helpSystem 208, 210, 400, 403  
 helpWindow 205, 206, 662, 697  
 hInstance 16-18, 85, 88, 259, 261, 262  
 hmq 156  
 hotKey 7, 118, 120, 280, 400, 401, 404,  
   408, 412, 416, 421, 427, 428, 609  
   stripping 609  
   text 609  
 HotKey (function) 427  
 HotKey (virtual function)  
   window object implementation of 427  
 hotkey designator  
   & 609  
 hotKeyMapTable 400,404  
 hour 162, 166, 715-728, 735, 736, 739,  
   740, 741, 742, 744, 748, 750, 751  
 hPrevInstance 16-18, 85, 88, 259, 261,  
   262  
 hps 271,272  
 hundredth 715, 716, 719, 722, 733, 734  
 hundredths 402, 465, 715, 717-720, 723,  
   724, 725-728, 733-735  
 hWnd 146, 148

## H

hab 85, 88  
 hardware 6, 59, 90, 199, 465, 473, 481,  
   495, 595, 604-607, 610-612, 614, 615,  
   649, 651-657  
   character set 605, 606, 611, 652, 653,  
     656  
 hardware character set  
   converting 607, 612, 655, 657  
   mapping 604

## I

il8n 4, 6, 11, 12, 144, 211, 370, 481, 512,  
   557, 558, 563, 564, 566, 568, 581,  
   582, 583-587, 589-592, 595, 617-619,  
   627, 628, 630, 631, 633, 634, 643,  
   644, 646, 647, 649, 650, 754  
 I18nInitialize (function) 601  
 il8nName 581, 582, 589, 591, 592  
 igrnum (number type) 499-502, 505, 506,  
   509, 510

- icharString 595, 602
- iconArray 37, 85, 86, 100, 101, 195, 245, 246, 251, 259, 265, 271, 375, 460
- Icon Array ToHandle (function) 100
- IconHandleToArray (function) 101
- IconHandleToArray (virtual function) 101
- iconHeight 37, 85, 86, 100, 101, 195, 245, 251, 259, 265, 271, 375, 460
- iconWidth 37, 85, 86, 100, 101, 195, 245, 251, 259, 265, 271, 375, 460
- Import (function) 509, 540, 721, 750
  - bignum implementation of 509
  - date implementation of 540
  - time implementation of 721
- inactive object 424
- include file
  - UI\_DSP.HPP 7
  - UI\_EVT.HPP 7
  - UI\_GEN.HPP 6
  - UI\_WIN.HPP 8
- index 117, 152, 198, 219, 228-231, 254, 378, 412, 763
- Index (function) 230
- Information (function) 28, 47, 74, 128, 185, 356, 394, 429
- Information (virtual function)
  - element implementation of 128
  - window manager implementation of 394
- Inherited (function) 432
  - window object implementation of 432
- initgraph( ) (function) 41
- initializing
  - graphics screen 40, 43, 198, 201, 248, 254, 256, 261, 262, 267-269, 273, 274, 378, 461
- initializing applications
  - UI\_APPLICATION 15
- initialState 63
- input
  - receiving 155
- input device 59
  - changing states 164
  - cursor 465
  - positioning of 162
  - programmer defined 60
  - reserved values for 146
  - states 61
- input information 145
  - mouse 483
  - position 295
  - region 327
- input management 155
- InputType (function) 152
- installed 41, 60, 61, 85, 87, 126, 371
- intCurrencySymbol 640
- interleaveStipple 85, 89
- internationalization 593
- interval 493-496
- intFractionDigits 639, 640
- inum 693, 694
- isForeground 37, 86, 104, 105, 195, 246, 251, 259, 265, 271, 308, 369, 375, 460
- isMono 85, 87
- IsNonSpacing (function) 601
- ISO8859-1 602, 603
- isoImageName 566
- isoLanguageName 630
- isoString 595, 602, 603
- ISOtOCHAR (function) 602
- ISOtOUNICODE (function) 603
- isText 85-87, 90, 95, 100, 107, 110, 113, 420
- item
  - definition of structure 213

**J**

- jday 739, 740, 753

**K**

- key 3, 8, 11, 13, 60-62, 145-149, 153, 164, 167, 176, 217, 218, 220, 239, 280, 391, 392, 404, 408, 422, 428, 434, 473-475, 483, 484
- keyboard 473



- break handler 473
- raw scan codes 474
- reading characters from 155
- shift state 474

**L**

- langName 593, 595, 608, 615
- language 4, 6, 11-13, 16, 139, 142, 144, 205, 206, 211, 569, 575, 581, 582, 589, 591-593, 601, 608, 615, 617-632, 640, 684, 694, 695, 740, 741, 754
- language data file
  - traversing 588
- languageName 139, 144, 205, 211, 593, 601, 627, 629, 631, 739, 753
- Last (function) 170, 186, 231, 288, 320, 349
- lastPalette 100, 107, 110, 282, 401, 412
- lastTime 442, 494
- Leap Year (function) 752
- length
  - multi-byte string 607, 654
- level 64, 65, 156, 186, 277, 318, 389, 395, 423, 430-432, 451, 499, 529, 564, 628, 644, 691, 715
- Line (function) 102
- Line (virtual function) 102
- Link (function) 664
- LinkMain (function) 21
- list 219
- list block 237
- list element 125
- ListIndex (function) 129
- lists
  - definition of 219
  - finding the next element 50, 67, 129, 438
  - finding the previous element 50, 69, 131, 440
  - list-block use of 317
  - setting the current item 232
- listScreenID 400, 405
- Load (function) 32, 53, 77, 189, 360, 453, 577, 586, 622, 687
  - window object implementation of 359, 453
- LoadDefaultDecorations (function) 565
- LoadDefaultI18N (function) 591
- LoadDefaultLanguage (function) 629
- LoadDefaultLocale (function) 645
- LoadCHARtoHardware (function) 604
- locale 4, 6, 11-13, 461, 500, 501, 506, 508, 509, 512, 557, 581, 582, 589, 591, 593, 595, 601, 608, 613-615, 633, 634-636, 639, 643-647, 740-742, 754, 755
  - initializing 608
- localeName 500, 512, 593, 601, 643, 645, 646, 647, 739, 754
- localization data 593
- logical mapping
  - of color palettes 281
  - of raw events 177
- logical messages
  - reserved values for 147
- logicalEvent 95, 100, 107, 110, 113, 178, 400, 425, 434-436, 442, 447
- LogicalEvent (function) 434
- logicalFont 38, 43, 196, 201, 252, 256, 272, 274, 275, 376, 380
- logicalPalette 110, 279-282, 400, 435, 436
- LogicalPalette (function) 435
- logicalValue 175, 176
- lParam 146, 148, 150
- lPort 307, 309
- lpszCmdLine 16-18, 262

**M**

- MACFONT 245-247
- machineName 595, 614
- MachineName (function) 614
- Main (function) 23
  - ULAPPLIC ATION 15
- MakeCanonical (function) 753
- MakeFullPath (function) 704

manage 73, 179, 219, 402, 444, 446, 473, 481  
 Manager (function) 48  
 map table 604-606, 611, 652, 653, 656  
     exception 604  
 MapChar (function) 605, 652  
 MapColor (function) 104  
 MapEvent (function) 177  
 mapName 595, 604, 605, 649-651  
 MapNSColor (function) 269  
 MapPalette (function) 281  
 mapped 93, 96, 101, 105, 175, 177, 250, 269, 281, 390, 595, 605, 606, 611, 612, 649, 652, 653, 656, 657  
 mapped text 605, 606, 611, 652, 653, 656  
 mapping  
     events 434  
     palettes 435  
     string 605, 606, 652, 653  
 MapRGB Color (function) 249  
 mapTable 175, 177, 279, 281  
 MapText  
     function 653  
 MapText (function) 605, 653  
 markPalette 85, 88  
 matchData 219, 228, 229  
 matchID 400,432,433  
 maxColors 38, 40, 196, 197, 246, 248, 252, 253, 260, 261, 266, 267, 272, 273, 376, 377  
 maxHeight 37, 39, 195, 196  
 maxSize 739, 748  
 maxWidth 37, 39, 195, 197  
 mblen (function) 606, 654  
 mbstowcs  
     function 654  
 mbstowcs (function) 607, 654  
 MDI windows 407  
 mdiChild 400,447,448  
 menu 5, 9, 11, 214, 247, 248, 267, 392, 401, 405, 407, 410, 443-445  
 menuBar 245, 248, 265, 267  
 menuScreenID 400, 405  
 messageField 205, 206  
 mevent 146, 148  
 Microsoft  
     mouse driver 481  
     Microsoft Windows  
         graphics display 259  
     military date formats 531, 542  
     millisecond 739-741, 744, 748, 750, 751  
     miniDenominatorX 85, 87, 88  
     miniDenominatorY 85, 88  
     minimum 71-73, 75, 179, 365, 541, 722, 750  
     miniNumeratorX 85, 87, 88  
     miniNumeratorY 85, 88  
     minute 715-724, 739-742, 744, 748, 750, 751  
     minutesWestGMT 593, 595  
     MkDir (function) 665  
     modifiers 145, 148, 175, 176, 474, 483, 485, 599  
     Modify (function) 30, 49, 75, 358, 436  
     modifyTime 669, 682, 693, 694  
     monDecimalSeparator 640  
     monGrouping 640  
     monoAttribute 277  
     monoBitmap 37, 85, 92, 95-98, 195, 245, 251, 259, 265, 271, 308, 375, 459, 460  
     month 529-532, 534-543, 739-744, 746, 748, 750, 751  
     monThousandsSeparator 640  
     MOTIFFONT (structure) 459, 460  
     mouse 481  
         position of screen 482  
         reading information from 155  
     MouseMove (function) 490  
     mpl 146, 148  
     mp2 146, 148  
     MSC\_FONT (structure) 252  
     MSCPATIERN 251-253  
     msec 494  
     msg 145, 146, 148, 483  
     multi-byte string  
         converting 607, 612, 655, 657  
         length 607, 654  
     multX 37, 38  
     multY 37, 38  
     myDayStrings 740, 741, 754  
     myDecorations 481, 491

myLanguage 139, 144, 205, 206, 211  
myLocale 500, 501, 509, 512, 740, 741,  
754  
myMonthStrings 740, 741, 754  
myTimedEvent 16, 17  
myTimeStrings 740, 741, 754

## N

nargs 402, 413  
nativeType 400, 434  
nCmdShow 16-18, 85, 88, 259, 261, 262  
NeedsUpdate (function) 438  
negativeSign 641  
negCurrencyPrecedes 639, 640  
negSignPrecedes 639, 641  
negSpaceSeparation 639, 641  
nevent 146, 149  
New (function) 33, 54, 78, 191, 361, 454  
    window object implementation of 454  
newColumn 38, 86, 114, 196, 246, 252,  
260, 266, 272, 369, 376, 460  
newElement 219, 223, 224  
newExtension 693, 698  
NewFunction (function) 34, 55, 80, 192,  
363, 456  
newLine 38, 86, 114, 196, 246, 252, 260,  
266, 272, 369, 376, 460  
newName 659, 665, 666, 669, 670, 694,  
699, 700  
newRegion 38, 86, 110-112, 115, 196,  
246, 252, 260, 266, 271, 369, 376,  
460  
newScreenID 38, 86, 115, 196, 246, 252,  
260, 266, 272, 369, 376, 460  
Next (function) 50, 67, 129, 293, 325,  
344, 438  
nextColor 265, 269  
NEXTFONT 265, 266, 307, 308, 310  
NextPathName (function) 288  
nmFlags 499, 505  
nObjectID 569, 570, 575, 576, 675, 676,  
683, 684  
noOfBitmapElements 557, 559

noOfElements 64, 155-157, 238-240, 317,  
318, 324, 617, 618  
noOfTextElements 557, 559  
notifyList 494, 498  
number  
    format 639  
    low-level 499  
    string representation 597  
NUMBER\_DECIMAL 499, 500, 507  
NUMBER\_WHOLE 499, 500, 507  
numberID 8, 25, 26, 45, 46, 48, 394, 400,  
401, 411, 426, 427, 430, 439, 440,  
527, 557, 559-561, 617, 619, 620,  
625, 713, 394, 430  
NumberID (function) 439  
numerator 353, 354, 356-358  
numOptions 459, 461  
numPoints 37, 86, 105, 106, 196, 246,  
251, 259, 265, 271, 308, 375, 460

## O

object retrieval  
    first in list 227, 320, 348  
    from a list 228  
    last in list 231, 320, 349  
    next in list 50, 68, 130, 439  
    previous in list 51, 70, 131, 440  
objectID 6, 8, 25, 28, 45-47, 51, 71, 74,  
125, 128, 175, 177-179, 185, 186,  
279, 281, 353, 356, 383, 394, 395,  
400, 401, 411, 429-436, 448, 449,  
569, 570, 575, 576, 666, 668, 675,  
676, 683-685, 700-702  
objectName 677, 685, 693, 698, 699, 707,  
708  
objectPathName 693, 707, 708  
objectTable 25, 31-36, 45, 52-57, 71, 76,  
77, 78-81, 179, 188-193, 353, 359,  
360, 361-364, 400, 401, 403, 412,  
452, 453-457, 403  
oemCountryCode 633  
oldObject 659, 666

- oldRegion 38, 86, 110-112, 114, 115, 196, 246, 252, 259, 260, 266, 271, 272, 369, 376, 460
- oldScreenID 38, 86, 115, 196, 246, 252, 260, 266, 272, 369, 376, 460
- OpenDir (function) 705
- operating system
  - character set 604
  - locale data 608
- operatingSystem 85, 86, 88, 90
- Operator != (function) 334, 522, 556, 737, 761
- Operator + (function) 173, 181, 223, 387, 514, 545, 726, 756
- Operator ++ (function) 335, 517, 551, 733
- Operator += (function) 553, 734
- Operator += (function) 337, 519
- Operator - (function) 174, 515, 546, 727, 757
- Operator - (function) 336, 518, 552, 733
- Operator -= (function) 338, 520, 554, 735
- Operator \* (function) 516
- Operator < (function) 525, 549, 731, 759
- Operator <= (function) 526, 550, 732, 760
- Operator = (function) 513, 544, 725, 755
- Operator == (function) 333, 521, 555, 736, 760
- Operator > (function) 523, 547, 729, 758
- Operator >= (function) 524, 548, 730, 758
- operator overload
  - != 522, 556, 737, 761
  - + 181, 223, 387, 514, 545, 726, 756
  - ++ 517, 551, 733
  - += 519, 553, 734
  - 234, 396, 515, 546, 727, 757
  - 518, 552, 733
  - = 520, 554, 735
  - \* 516
  - < 525, 549, 731, 759
  - <= 526, 550, 732, 760
  - = 513, 544, 725, 755
  - == 521, 555, 736, 760
  - > 523, 547, 548, 729, 730, 758
  - >= 524, 548, 730, 758
- operator overloads
  - != 334
  - ++ 335
  - 336
  - = 333
- Operator - (function) 234, 396
- options 135-137, 143, 251, 252, 312, 315, 375, 376, 459, 461, 462, 532, 537, 542
- ordering
  - printf arguments 596
- OS/2
  - graphics display 271
- os2ClassName 444
- OSI18nInitialize (function) 608
- Overlap (function) 330
- overloaded operators
  - += 337
  - = 338

**P**

- packedDate 529, 530, 535, 540
- packedTime 715, 716, 719, 722
- palette
  - definition structure 277
- palette mapping 435
- paletteMapTable 110, 282, 400, 404, 459
- palettes
  - logical mapping 279
- ParseLangEnv (function) 615
- pasteBuffer 401, 412
- pasteLength 401, 412
- path
  - data file 283
  - environment variable 283
- pathLen 694, 703
- pathName 287, 289, 291, 292, 581, 582, 661, 677, 685, 693, 696-699, 707, 708
- paths
  - creating 697
  - finding 287, 288
  - splitting 707
  - valid recognition 709
- patterns 247, 261, 377

patternTable 39, 197, 198, 247, 253, 254,  
     261, 377, 378  
 pDay 740, 743  
 pDayOfWeek 740, 743  
 persistent objects  
     New (function) 403  
 pFlags 569, 570, 659, 660, 675, 676  
 pixMapColorTable 245, 247  
 places 267, 273, 405, 411, 479, 491, 498,  
     499, 500, 507, 511-513, 566, 630, 646  
 pMode 308, 312, 314, 315  
 pMonth 740, 743  
 point 29, 48, 103, 106, 144, 211, 214,  
     220, 229, 246, 295, 296, 309, 333,  
     393, 416, 425, 462, 498, 499, 501,  
     506, 507, 510, 512, 513, 640, 664,  
     754  
 pointer device  
     changing images 160  
 pointSize 307, 309  
 Poll (function) 68, 471, 479, 491  
 Poll (virtual function)  
     cursor implementation of 471  
     device implementation of 68  
     keyboard implementation of 479  
     mouse implementation of 491  
     timer implementation of 498  
 Polygon (function) 105  
 Polygon (virtual function) 105  
 polygonPoints 38, 86, 105, 196, 246, 251,  
     259, 265, 271, 308, 375, 460  
 posCurrencyPrecedes 639, 640  
 position indicator 295  
     cursor 465  
 positiveSign 640  
 posSignPrecedes 639, 640  
 posSpaceSeparation 639, 640  
 POSTSCRIPTFONT 307, 309  
 postSpace 85, 87, 442  
 preSpace 85, 87, 442  
 Previous (function) 50, 69, 131, 294, 325,  
     345, 440  
 PRINTERFONT 307, 310  
 printerMode 307, 309, 312, 314  
 printerPort 307, 309  
 printf-type functions

    enhancements 596  
 prinJob 307, 310  
 processError 401, 451  
 proclInstance 401, 444  
 program termination 391  
 programPath 283, 284, 292  
 psFontTable 309  
 Put (function) 171, 172  
 pwcs 595, 607, 612, 649, 654, 657  
 pYear 740, 743

## Q

qFlags 156, 172  
 QFlags (function) 172  
 queueBlock 64, 156, 240-242, 244, 318,  
     319, 324

## R

range 76, 365, 366, 406, 409, 451, 509,  
     510, 531, 536, 540, 541, 716, 717,  
     719, 722, 723, 739, 742, 750, 751  
 rawCode 145, 147, 148, 175-177, 218,  
     296, 332, 437, 474, 483, 484  
 rbignum (number type) 499, 501, 502,  
     505, 506, 509-511  
 ReadDir (function) 672  
 reading  
     from keyboard 168  
     from mouse 168  
 rect 327, 328  
 Rectangle (function) 107  
 Rectangle (virtual function) 107  
 RectangleXORDiff (function) 110  
 RectangleXORDiff (virtual function) 110  
 recurse 740  
 Redisplay Type (function) 440  
 refNumberID 25, 26  
 RegionConvert (function) 441  
 RegionDefine (function) 112

RegionDefine (virtual function) 112  
 RegionInitialize (function) 114  
 RegionMax (function) 442  
 RegionMove (function) 114  
 RegionMove (virtual function) 114  
 RegisterObject 444  
 RegisterObject (function) 444  
 rememberCWD 283, 284, 292  
 removing hotkey characters 609  
 removing windows 392  
 RenameObject (function) 666  
 repeatRate 399, 402  
 ReportError (function) 136  
 request 6, 8, 9, 25, 28-30, 45, 47, 48, 71,  
     74, 75, 105, 125, 128, 179, 185, 186,  
     353, 356-358, 383, 394, 395, 400,  
     429, 430-432, 592  
 reserved values  
     input devices 146  
     logical messages 147  
     system messages 146  
 retValue 595, 603  
 revision 693, 694  
 revisions 659, 668-670  
 RewindDir (function) 672  
 rgbColorMap 245, 247  
 rlcFlags 353, 354, 357  
 Rmdir (function) 667  
 Root (function) 447  
 round (function)  
     bignum implementation of 511, 512

## S

SampleFunction (function) 2  
 Save (function) 668  
 SaveAs (function) 669  
 screen 83  
     coordinates 87  
     regions 341, 350  
 screen colors 281  
 screen identification 84  
 ScreenDump (function) 314

scroll 3, 5, 8, 9, 11, 13, 111, 145, 148,  
     150, 217, 365, 366, 402, 405, 407,  
     474, 475, 483, 484  
 scroll bar  
     position 365  
 scrolling  
     scroll bar 365  
 search path 284  
 searchID 45, 46, 51, 400, 401, 403, 411,  
     448, 449  
 SearchID (function) 51, 448  
 searchPath 15-18, 20, 23, 37, 39, 195,  
     197, 251, 253, 375, 377, 694, 695  
 searchPath (static variable) 695  
 Seek (function) 689  
 SeekDir (function) 673  
 SetBasis (function) 543  
 SetCTime (function) 678  
 SetCurrent (function) 232  
 SetDecorations (function) 566  
 SetFont (function) 43, 201, 256, 274, 380  
 SetLanguage (function) 144, 211, 630, 753  
 SetLocale (function) 512, 646, 754  
 SetMTime (function) 679  
 SetPattern (function) 43, 201, 256, 380  
 shell 89,401,411  
 shiftState 8, 147, 217, 218, 474  
 showing 365, 366  
 signStr 499, 505  
 signString 499, 501, 509  
 size  
     multi-byte character 607, 654  
 Sort (function) 233  
     list implementation of 233  
 spacing  
     character 602  
 Split (function) 350  
 src 595, 597, 598  
 startAngle 37, 85, 98, 99, 195, 245, 251,  
     259, 265, 271, 308, 375, 460  
 static variables  
     eventMapTable 112, 178, 400, 404  
 Stats (function) 689, 705  
     storage implementation of 705  
 status  
     general window objects 409

- storage
  - setting default 403
- Storage (function) 690
- storage files 659, 693
- storage objects 675, 683
- storageError (variable) 661, 665, 667, 669, 694, 695, 696, 700, 695
- StorageName (function) 706
- Store (function) 35, 56, 80, 193, 363, 456, 571, 587, 623, 680, 690
- Store (virtual function) 35, 56, 80, 193, 363, 456
- str 594, 595, 600, 613
- strcmp
  - wild card characters 613
- string
  - converting 602, 603, 607, 612, 655, 657
  - filename 598
  - ISO 602, 603
  - manipulation 595
  - multi-byte 607, 612, 655, 657
  - wide-character 607, 612, 655, 657
- stringID 395, 400, 411, 427, 431, 439, 449, 527, 625, 666, 683, 684, 701, 702, 703, 713, 395, 431
- StringID (function) 449
- strip 415, 416
- StripFullPath (function) 707
- StripHotMark (function) 609
- Strstrip (function) 609
- Subtract (function) 172, 187, 234, 243, 396
  - list-block implementation of 243
- SVGA mode 199
- SwapData (function) 625
- system messages
  - reserved values for 146

## T

- Tell (function) 691
- TellDir (function) 674
- tempname 693, 708
- TempName (function) 708

- text
  - determining height 117
  - determining width 119
  - mapped 605, 606, 611, 652, 653, 656
  - palette mapping 279
  - palettes 277
  - presentation of 116
  - unmapped 611, 656
- Text (function) 116
- Text (virtual function) 116
- text display 369
- text mode
  - border 372
  - characters 372
- TextHeight (function) 117
- TextHeight (virtual function) 117
- textScreenID 400, 405
- TextWidth (function) 119
- Text Width (virtual function) 119
- thousandsSeparator 640
- time
  - format 641
  - low-level 715
- time\_t 694
- time\_12StringFormat 639, 641
- timer device 493
- times
  - alphanumeric 717, 718, 721
  - format flags 717
  - system 716
- timeSeparator 641
- TimeStamp (function) 610
- timeStringFormat 639, 641
- title 5, 8, 10, 21, 22, 24, 39, 130, 131, 135, 137, 139, 142, 143, 178, 197, 205, 206, 215, 224, 253, 260, 267, 272, 377, 388, 401-403, 407, 444-446, 461
- titleField 205, 206
- titleMessage 133, 135, 136, 139, 142
- tmFlags 715, 716, 719, 722
- tmppath 694, 704
- tmrFlags 493-495
- topShell 85, 89
- TopWidget (function) 450
- toStandard 649, 650

Touch (function) 682  
Touching (function) 332  
Traverse (function) 587  
traversing  
    data file 588  
truncate (function)  
    bignum implementation of 512  
typeFace 246, 251, 252, 307, 309, 375,  
    376

## U

U.S. date formats 532, 543  
UI\_APPLICATION (class) 15  
UI\_APPLICATION (function) 18  
UI\_ATTACHMENT (class) 25  
UI\_ATTACHMENT (function) 26  
UI\_BGI\_DISPLAY (class) 37  
UI\_BGI\_DISPLAY (function) 40  
UI\_CONSTRAINT (class) 45  
UI\_CONSTRAINT (function) 46  
UI\_DEVICE (class) 59  
UI\_DEVICE (function) 61  
UI\_DIMENSION\_CONSTRAINT (class)  
    71  
UI\_DIMENSION\_CONSTRAINT  
    (function) 72  
UI\_DISPLAY (class) 83  
UI\_DISPLAY (function) 90  
UI\_ELEMENT (class) 125  
UI\_ELEMENT (function) 126  
UI\_ERROR\_STUB (class) 133  
UI\_ERROR\_SYSTEM (class) 139  
UI\_ERROR\_SYSTEM (function) 140  
UI\_EVENT (function) 148  
UI\_EVENT (structure) 145  
UI\_EVENT\_MANAGER (class) 155  
UI\_EVENT\_MANAGER (function) 157  
UI\_EVENT\_MAP (structure) 175  
UI\_GEOMETRY\_MANAGER (class) 179  
UI\_GEOMETRY\_MANAGER (function)  
    180  
UI\_GRAPHICS\_DISPLAY (class) 195  
UI\_GRAPHICS\_DISPLAY (function) 198  
UI\_HELP\_STUB (class) 203  
UI\_HELP\_SYSTEM (class) 205  
UI\_HELP\_SYSTEM (function) 207  
UI\_ITEM (structure) 213  
UI\_KEY (structure) 217  
UI\_LIST (class) 219  
UI\_LIST (function) 221  
UI\_LIST\_BLOCK (class) 237  
UI\_LIST\_BLOCK (function) 239  
UI\_MACINTOSH\_DISPLAY (class) 245  
UI\_MACINTOSH\_DISPLAY (function)  
    248  
UI\_MSC\_DISPLAY (class) 251  
UI\_MSC\_DISPLAY (function) 254  
UI\_MSWINDOWS\_DISPLAY (class) 259  
UI\_MS\_WINDOWS\_DISPLAY (function)  
    261  
UI\_NEXTSTEP\_DISPLAY (class) 265  
UI\_NEXTSTEP\_DISPLAY (function) 267  
UI\_OS2\_DISPLAY (class) 271  
UI\_OS2\_DISPLAY (function) 273  
UI\_PALETTE (structure) 277  
UI\_PALETTE\_MAP (structure) 279  
UI\_PATH (class) 283  
UI\_PATH (function) 284  
UI\_PATH\_ELEMENT (class) 291  
UI\_PATH\_ELEMENT (function) 291  
UI\_POSITION (structure) 295  
UI\_PRINTER (class) 307  
UI\_PRINTER (function) 310  
UI\_QUEUE\_BLOCK (class) 317  
UI\_QUEUE\_BLOCK (function) 317  
UI\_QUEUE\_ELEMENT (class) 323  
UI\_QUEUE\_ELEMENT (function) 323  
UI\_REGION (structure) 327  
UI\_REGION\_ELEMENT (class) 341  
UI\_REGION\_ELEMENT (function) 342  
UI\_REGION\_LIST (class) 347  
UI\_RELATIVE\_CONSTRAINT (class)  
    353  
UI\_RELATIVE\_CONSTRAINT (function)  
    354  
UI\_SCROLL\_INFORMATION (structure)  
    365  
UI\_TEXT\_DISPLAY (class) 369  
UI\_TEXT\_DISPLAY (function) 370



- UI\_WCC\_DISPLAY (class) 375
- UI\_WCC\_DISPLAY (function) 378
- UI\_WINDOW\_MANAGER (class) 383
- UI\_WINDOW\_MANAGER (function) 385
- UI\_WINDOW\_OBJECT (class) 399
- UI\_WINDOW\_OBJECT (function) 413, 452
- UI\_XT\_DISPLAY (class) 459
- UI\_XT\_DISPLAY (function) 461
- UID\_CURSOR (class) 465
- UID\_CURSOR (function) 466
- UID\_KEYBOARD (class) 473
- UID\_KEYBOARD (function) 475
- UID\_MOUSE (class) 481
- UID\_MOUSE (function) 485
- UID\_TIMER (class) 493
- UID\_TIMER (function) 495
- Unicode** 90, 195, 197, 459, 461, 593, 595, 598, 599, 600, 602-607, 611, 612, 630, 649, 651, 654, 655, 657
  - character set 607, 655
  - converting 607, 612, 655, 657
  - converting to 602, 603
  - mapping 604-606, 652, 653
  - unmapping 611, 656
  - wild card characters 613
- UnMapChar (function) 610, 655
- unmapped 595, 605, 606, 611, 612, 649, 653, 655-657
- unmapped text 605, 606, 611, 652, 653, 656
- unmapping
  - string 611, 656
- UnMapText (function) 611, 656
- useArgs 402, 444, 446
- usec 739, 740, 753
- useCount 565, 567, 581, 582, 590, 592, 629, 631, 645, 647, 693, 694
- UseDecorations (function) 567
- useDefault 557, 559-561, 617, 619, 620
- usedMenuID 247
- UseI18N (function) 591
- UseLanguage (function) 631
- UseLocale (function) 647
- user events 147
- user function

- item use of 214
- parameters 410
- userFlags 400, 410
- userFunction 400, 410-412, 414, 450, 451
- UserFunction (function) 450
- userFunctionName 401, 404, 412
- userObject 400, 410, 412
- userObjectName 401, 412
- userStatus 400, 410
- userTable 25, 31-36, 45, 52-57, 71, 76-81, 179, 188-194, 353, 359-364, 400, 401, 404, 410, 412, 452, 453-457
- utime 3, 6, 13, 494, 529, 715, 739-761

## V

- val 595, 599, 675, 678, 679
- Validate (function) 451
- Validate (virtual function)
  - window object implementation of 451
- validation 451
- ValidName (function) 709
- values
  - comparing dates 547-550, 555, 556
  - comparing integers 522
  - comparing numbers 521, 523-526
  - comparing times 729-732, 736, 737
- Version (function) 710
- virtual destructor 127
- virtual member functions
  - window object use of 399
- VirtualGet (function) 120
- VirtualGet (virtual function) 120
- VirtualPut (function) 122
- VirtualPut (virtual function) 122

## W

- WCCFONT 375-377
- WCCPATTERN 375-377
- Wcstombs (function) 612, 657

- wide-character string
  - converting 607, 612, 655, 657
- widgetClass 402, 444, 446
- width 37-39, 71, 73, 86-88, 93, 96, 98,
  - 101, 102, 103, 107-109, 112, 118-120,
  - 127, 195-197, 246, 251, 252, 259,
  - 265, 266, 271, 308, 327, 333, 369,
  - 375, 376, 402, 413, 414, 460
- Width (function) 333
- wild card characters
  - strcmp 613
- WildStrcmp (function) 613
- winClassName 401,444,445
- window characters
  - text mode 372
- window management 383
- window manager
  - adding windows 387
  - subtracting windows 392
  - UI\_APPLICATION 15
- window object 399
- windowID 282,411,433
- windowingSystem 85, 86, 88, 90, 91
- windowObject 145, 148, 498
- windows
  - cascading 392
  - centering 388
  - removing 396
- windowScreenID 400, 405
- WinMain (function)
  - UI\_APPLICATION 15
- wMsg 146, 148
- woAdvancedFlags (variable) 127, 282,
  - 400, 402, 407, 413, 414, 407
- woFlags 95, 113, 127, 280, 400, 402, 405,
  - 413, 414, 417, 442
- woFlags (variable) 405
- woStatus 280, 400, 409, 417, 433
- woStatus (variable) 409
- wParam 146, 148

## X

- X 6, 84, 89, 106, 218, 459-462

## Index

- xDisplay 85, 89
- xevent 145,146,148,151,483
- xGc 85,89
- xorGC 85, 89
- xorPalette 85, 88, 282
- xRadius 37, 85, 98, 195, 245, 251, 259,
  - 265, 271, 308, 375, 460
- xScreen 85, 89
- xScreenNumber 85, 89

## Y

- year 522, 524, 526, 529-532, 535-544,
  - 739, 740-744, 747, 748, 750-752
- yRadius 37, 85, 98, 195, 245, 251, 259,
  - 265, 271, 308, 375, 460

## Z

- ZIL\_BIGNUM (class) 499
- ZIL\_BIGNUM (function) 501
- ZIL\_BITMAP\_ELEMENT (class) 527
- ZIL\_DATE (class) 529
- ZIL\_DATE (function) 530
- ZIL\_DECORATION (class) 557
- ZIL\_DECORATION (function) 558
- ZIL\_DECORATION\_MANAGER (class)
  - 563
- ZIL\_DECORATION\_MANAGER
  - (function) 563
- ZIL\_DELTA\_STORAGE\_OBJECT (class)
  - 569
- ZIL\_DELTA\_STORAGE\_OBJECT\_READ
  - ONLY (class) 575
- ZIL\_DIGITS 499,500
- ZIL\_I18N (class) 581
- ZIL\_I18N (function) 582
- ZIL\_I18N\_MANAGER (class) 589
- ZIL\_INTERNATIONAL (class) 593
- ZIL\_LANGUAGE (class) 617
- ZIL\_LANGUAGE (function) 618, 620

ZIL\_LANGUAGE\_ELEMENT (class) 625  
 ZIL\_LANGUAGE\_MANAGER (class) 627  
 ZIL\_LANGUAGE\_MANAGER (function) 627  
 ZIL\_LOCALE (class) 633  
 ZIL\_LOCALE (function) 634  
 ZIL\_LOCALE\_ELEMENT (class) 639  
 ZIL\_LOCALE\_MANAGER (class) 643  
 ZIL\_LOCALE\_MANAGER (function) 643  
 ZIL\_MAP\_CHARS (class) 649  
 ZIL\_MAP\_CHARS (function) 650  
 ZIL\_NUMBER 499,501  
 ZIL\_STATS\_INFO 13, 683, 689, 690, 693, 694, 705, 706  
 ZIL\_STORAGE (class) 659  
 ZIL\_STORAGE (function) 660  
 ZIL\_STORAGE (function) 684  
 ZIL\_STORAGE\_DIRECTORY (class) 671  
 ZIL\_STORAGE\_OBJECT (class) 675  
 ZIL\_STORAGE\_OBJECT (function) 676  
 ZIL\_STORAGE\_OBJECT\_READ\_ONLY (class) 683  
 ZIL\_STORAGE\_READ\_ONLY (class) 693  
 ZIL\_STORAGE\_READ\_ONLY (function) 695  
 ZIL\_TEXT\_ELEMENT (structure) 713  
 ZIL\_TIME (class) 715  
 ZIL\_TIME (function) 716  
 ZIL\_IJTIME (class) 739  
 ZIL\_IJTIME (function) 741  
 OpenZinc events 146  
 OpenZinc graphics 195  
 ZINC\_LANG 615  
 ZINC\_PATH environment variable 283  
 ZINCFONT 197  
 zoneOffset 740, 741  
  
 ~UI\_APPLICATION (function) 20  
 ~UI\_ATTACHMENT (function) 28  
 ~UI\_BGI\_DISPLAY (function) 42  
 ~UI\_CONSTRAINT (function) 47  
  
 ~UI\_DEVICE (function) 63  
 ~UI\_DIMENSION\_CONSTRAINT (function) 73  
 ~UI\_DISPLAY (function) 91  
 ~UI\_ELEMENT (function) 127  
 ~UI\_ERROR\_STUB (function) 133  
 ~UI\_ERROR\_SYSTEM (function) 141  
 ~UI\_EVENT\_MANAGER (function) 158  
 ~UI\_GEOMETRY\_MANAGER (function) 181  
 ~UI\_GRAPHICS\_DISPLAY (function) 200  
 ~UI\_HELP\_STUB (function) 203  
 ~UI\_HELP\_SYSTEM (function) 208  
 ~UI\_LIST (function) 222  
 ~UI\_LIST\_BLOCK (function) 240  
 ~UI\_MACINTOSH\_DISPLAY (function) 249  
 ~UI\_MSC\_DISPLAY (function) 255  
 ~UI\_MSWINDOWS\_DISPLAY (function) 262  
 ~UI\_NEXTSTEP\_DISPLAY (function) 268  
 ~UI\_OS2\_DISPLAY (function) 274  
 ~UI\_PATH (function) 285  
 ~UI\_PATH\_ELEMENT (function) 293  
 ~UI\_QUEUE\_BLOCK (function) 318  
 ~UI\_QUEUE\_ELEMENT (function) 324  
 ~UI\_REGION\_ELEMENT (function) 343  
 ~UI\_RELATIVE\_CONSTRAINT (function) 356  
 ~UI\_TEXT\_DISPLAY (function) 372  
 ~UI\_WCC\_DISPLAY (function) 379  
 ~UI\_WINDOW\_MANAGER (function) 386  
 ~UI\_WINDOW\_OBJECT (function) 414  
 ~UI\_XT\_DISPLAY (function) 463  
 ~UID\_CURSOR (function) 467  
 ~UID\_KEYBOARD (function) 476  
 ~UID\_MOUSE (function) 486  
 ~UID\_TIMER (function) 496  
 ~ZIL\_BIGNUM (function) 503  
 "ZIL\_I18N (function) 583  
 ~ZIL\_IVIAP\_CHARS (function) 651  
 ~ZIL\_STORAGE (function) 661  
 ~ZIL\_STORAGE\_ (function) 686

~ZIL\_STORAGE\_DIRECTORY (function)  
671  
~ZIL\_STORAGE\_OBJECT (function) 678  
~ZIL\_STORAGE\_READ\_ONLY (function)  
696  
~ZIL\_UTIME (function) 743

GNU Free Documentation License  
Version 1.3, 3 November 2008

Copyright (C) 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

[<http://fsf.org/>](http://fsf.org/)

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical

connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a

section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

## 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

#### 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers



- or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
  - N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
  - O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections

Entitled "Endorsements".

## 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual

title.

## 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

## 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

## 11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A

public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.