TColor class

Header File
color.h

Description
TColor is a support class used in conjunction with the classes *TPalette*, *TPaletteEntry*, *TRgbQuad*, and TRgbTriple to simplify all color operations. *TColor* has ten static data members representing the standard RGB COLORREF values, from *Black* to *White*. Constructors are provided to create *TColor* objects from COLORREF and RGB values, palette indexes, palette entries, and RGBQUAD and RGBTRIPLE values.

See the entries for *NBits* and *NColors* for a description of *TColor*-related functions.

Public Data Members

- Black
- Gray
- LtBlue
- LtCyan
- LtGray
- LtGreen
- LtMagenta
- LtRed
- LtYellow
- None
- Sys3dDkShadow
- Sys3dFace
- Sys3dHilight
- Sys3dLight
- Sys3dShadow
- SysActiveBorder
- SysActiveCaption
- SysAppWorkspace
- SysBtnText
- SysCaptionText
- SysDesktop
- SysGrayText
- SysHighlight
- SysHighlightText
- SysInactiveBorder
- SysInactiveCaption
- SysInactiveCaptionText
- SysInfoBk
- SysInfoText
- SysMenu
- SysMenuText
- SysScrollbar
- SysWindow
- SysWindowFrame

SysWindowText
Transparent
White

Public Constructors

TColor::TColor

Public Member Functions

Blue
Flags
GetValue
GetSysColor
Green
Index
IsSysColor
operator ==
operator COLORREF
operator =
operator !=
PalIndex
PalRelative
Red
Rgb
SetSysColors
SetValue

Private Data Members

Value

TColor::Black

TColor

Syntax

```
static const TColor Black;
```

Description

The static *TColor* object with fixed *Value* set by RGB(0, 0, 0).

TColor::Gray

TColor

Syntax

```
static const TColor Gray;
```

Description

Contains the static *TColor* object with fixed *Value* set by RGB(128, 128, 128).

TColor::LtBlue

TColor

Syntax

```
static const TColor LtBlue;
```

Description

Contains the static *TColor* object with the fixed *Value* set by RGB(0, 0, 255).

TColor::LtCyan

TColor

Syntax

```
static const TColor LtCyan;
```

Description

Contains the static *TColor* object with the fixed *Value* set by RGB(0, 255, 255).

TColor::LtGray

TColor

Syntax

```
static const TColor LtGray;
```

Description

Contains the static *TColor* object with the fixed *Value* set by RGB(192, 192, 192).

TColor::LtGreen

TColor

Syntax

```
static const TColor LtGreen;
```

Description

Contains the static *TColor* object with the fixed *Value* set by RGB(0, 255, 0).

TColor::LtMagenta

TColor

Syntax

```
static const TColor LtMagenta;
```

Description

Contains the static *TColor* object with the fixed *Value* set by RGB(255, 0, 255).

TColor::LtRed

TColor

Syntax

```
static const TColor LtRed;
```

Description

Contains the static *TColor* object with the fixed *Value* set by RGB(255, 0, 0).

TColor::LtYellow

TColor

Syntax

```
static const TColor LtYellow;
```

Description

Contains the static *TColor* object with the fixed *Value* set by RGB(255, 255, 0).

TColor::None

TColor

Syntax

```
static const TColor None;
```

Description

These are special marker colors, using flag bit pattern. Value is never really used. Value must not change for streaming compatibility with OWL's TWindow.

TColor::Sys3dDkShadow

TColor

Syntax

```
static const TColor Sys3dDkShadow;
```

Description

For Windows 95 only. The symbolic system color value for dark shadow regions of 3-dimensional display elements. Performs GetSysColor() on conversion to COLORREF.

TColor::Sys3dFace

TColor

Syntax

```
static const TColor Sys3dFace;
```

Description

For Windows 95 only. The symbolic system color value for the face color of 3-dimensional display elements. Performs GetSysColor() on conversion to COLORREF.

TColor::Sys3dHilight

TColor

Syntax

```
static const TColor Sys3dHilight;
```

Description

For Windows 95 only. The symbolic system color value for highlighted 3-dimensional display elements (for edges facing the light source). Performs GetSysColor() on conversion to COLORREF.

TColor::Sys3dLight

TColor

Syntax

```
static const TColor Sys3dLight;
```

Description

For Windows 95 only. The symbolic system color value for the light color for 3-dimensional display elements (for edges facing the light source). Performs GetSysColor() on conversion to COLORREF.

TColor::Sys3dShadow

TColor

Syntax

```
static const TColor Sys3dShadow;
```

Description

For Windows 95 only. The symbolic system color value for the shadow regions of 3-dimensional display elements (for edges facing away from the light source). Performs GetSysColor() on conversion to COLORREF.

TColor::SysActiveBorder

TColor

Syntax

```
static const TColor SysActiveBorder;
```

Description

The symbolic system color value for the borders of the active window. Performs GetSysColor() on conversion to COLORREF.

TColor::SysActiveCaption

TColor

Syntax

```
static const TColor SysActiveCaption;
```

Description

The symbolic system color value for the caption of the active window. Performs GetSysColor() on conversion to COLORREF.

TColor::SysAppWorkspace

TColor

Syntax

```
static const TColor SysAppWorkspace;
```

Description

The symbolic system color value for the background of multiple document interface (MDI) applications. Performs GetSysColor() on conversion to COLORREF.

TColor::SysBtnText

TColor

Syntax

```
static const TColor SysBtnText;
```

Description

The symbolic system color value for the text on buttons. Performs GetSysColor() on conversion to COLORREF.

TColor::SysCaptionText

TColor

Syntax

```
static const TColor SysCaptionText;
```

Description

The symbolic system color value for text in captions and size boxes, and for the arrow boxes on scroll bars. Performs GetSysColor() on conversion to COLORREF.

TColor::SysDesktop

TColor

Syntax

```
static const TColor SysDesktop;
```

Description

The symbolic system color value for the desktop. Performs GetSysColor() on conversion to COLORREF.

TColor::SysGrayText

TColor

Syntax

```
static const TColor SysGrayText;
```

Description

The symbolic system color value for grayed (disabled) text. Performs GetSysColor() on conversion to COLORREF.

This color is set to 0 if the current display driver does not support a solid gray color.

TColor::SysHighlight

TColor

Syntax

```
static const TColor SysHighlight;
```

Description

The symbolic system color value for items selected in a control. Performs GetSysColor() on conversion to COLORREF.

TColor::SysHighlightText

TColor

Syntax

```
static const TColor SysHighlightText;
```

Description

The symbolic system color value for text selected in a control. Performs GetSysColor() on conversion to COLORREF.

TColor::SysInactiveBorder

TColor

Syntax

```
static const TColor SysInactiveBorder;
```

Description

The symbolic system color value for the borders of every inactive window. Performs GetSysColor() on conversion to COLORREF.

TColor::SysInactiveCaption

TColor

Syntax

```
static const TColor SysInactiveCaption;
```

Description

The symbolic system color value for the caption background of every inactive window. Performs GetSysColor() on conversion to COLORREF.

TColor::SysInactiveCaptionText

TColor

Syntax

```
static const TColor SysInactiveCaptionText;
```

Description

The symbolic system color value for the caption text of every inactive window. Performs GetSysColor() on conversion to COLORREF.

TColor::SysInfoBk

TColor

Syntax

```
static const TColor SysInfoBk;
```

Description

For Windows 95 only. The symbolic system color value for the background of tooltip controls. Performs GetSysColor() on conversion to COLORREF.

TColor::SysInfoText

TColor

Syntax

```
static const TColor SysInfoText;
```

Description

For Windows 95 only. The symbolic system color value for text shown on tooltip controls. Performs GetSysColor() on conversion to COLORREF.

TColor::SysMenu

TColor

Syntax

```
static const TColor SysMenu;
```

Description

The symbolic system color value for the background of menus. Performs GetSysColor() on conversion to COLORREF.

TColor::SysMenuText

TColor

Syntax

```
static const TColor SysMenuText;
```

Description

The symbolic system color value for the text shown on menus. Performs GetSysColor() on conversion to COLORREF.

TColor::SysScrollbar

TColor

Syntax

```
static const TColor SysScrollbar;
```

Description

The symbolic system color value for what is usually the gray area of scrollbars. This is the region that the scrollbar slider slides upon. Performs GetSysColor() on conversion to COLORREF.

TColor::SysWindow

TColor

Syntax

```
static const TColor SysWindow;
```

Description

The symbolic system color value for the background of each window. Performs GetSysColor() on conversion to COLORREF.

TColor::SysWindowFrame

TColor

Syntax

```
static const TColor SysWindowFrame;
```

Description

The symbolic system color value for the frame around each window. The frame is not the same as the border. Performs GetSysColor() on conversion to COLORREF.

TColor::SysWindowText

TColor

Syntax

```
static const TColor SysWindowText;
```

Description

The symbolic system color value for text in every window. Performs GetSysColor() on conversion to COLORREF.

TColor::Transparent

TColor

Syntax

```
static const TColor Transparent;
```

Description

Non-painting marker color using the flag bit pattern. This value is never really used. This value must not change, for streaming compatibility with OWL's TWindow.

TColor::White

TColor

Syntax

```
static const TColor White;
```

Description

Contains the static *TColor* object with the fixed *Value* set by RGB(255, 255, 255).

TColor::TColor

See also TColor

Syntax 1

```
TColor();
```

Description

The default constructor sets *Value* to 0.

Syntax 2

```
TColor(COLORREF value);
```

Description

Creates a *TColor* object with *Value* set to the given value.

Syntax 3

```
TColor(long value);
TColor(long value) : Value((COLORREF)value) {}
```

Description

Creates a *TColor* object with *Value* set to (COLORREF)*value*.

Syntax 4

```
TColor(int r, int g, int b);
```

Description

Creates a *TColor* object with *Value* set to RGB(*r*,*g*,*b*).

Syntax 5

```
TColor(int r, int g, int b, int f);
```

Description

Creates a *TColor* object with *Value* set to RGB(*r*,*g*,*b*) with the flag byte formed from *f*.

Syntax 6

```
TColor(int index);
```

Description

Creates a *TColor* object with *Value* set to *PALETTEINDEX(index)*.

Syntax 7

```
TColor(const PALETTEENTRY far& pe);
```

Description

Creates a *TColor* object with *Value* set to:

```
RGB(pe.peRed, pe.peGreen, pe.peBlue)
```

Syntax 8

```
TColor(const RGBQUAD far& q);
```

Description

Creates a *TColor* object with *Value* set to:

```
RGB(q.rgbRed, q.rgbGreen, q.rgbBlue)
```

Syntax 9

```
TColor(const RGBTRIPLE far& t);
```

Description

Creates a *TColor* object with *Value* set to:

```
RGB(t.rgbtRed, t.rgbtGreen, t.rgbtBlue)
```

TColor::Blue

See also TColor

Syntax

```
uint8 Blue() const;
```

Description

Returns the blue component of this color's *Value*.

TColor::Flags

See also TColor

Syntax

```
uint8 Flags() const;
```

Description

Returns the *peFlags* value of this object's *Value*.

TColor::GetSysColor

TColor

Syntax

```
static TColor GetSysColor(int uiElement);
```

Description

(Presentation Manager only) Returns the color of the given *uiElement*.

TColor::GetValue

TColor

Syntax

```
COLORREF GetValue() const;
```

Description

Gets a 32bit COLORREF type from this color object. Performs a GetSysColor() lookup if the object represents a symbolic sys-color index.

TColor::Green

See also TColor

Syntax

```
uint8 Green() const;
```

Description

Returns the green component of this color's *Value*.

TColor::IsSysColor

See also TColor

Syntax

```
bool IsSysColor() const;
```

Description

Returns true if the color is a system color, false otherwise.

TColor::operator ==

See also TColor

This function compares between two binary representation of colors; it does not compare colors logically.

For example, if palette entry 4 is solid red (rgb components (255, 0, 0)), the following will return false:

```
if (TColor(4) == TColor(255, 0, 0))
```

Syntax 1

```
bool operator ==(const TColor& other) const;
```

Description

Returns true if two colors are equal.

Syntax 2

```
bool operator ==(COLORREF cr) const;
```

Description

Returns true if this color matches a COLORREF.

TColor::operator =

TColor

Syntax

```
TColor& operator =(const TColor& src);
```

Description

Sets the value of color after it has been initialized.

TColor::operator !=

TColor

This function compares between two binary representation of colors; it does not compare colors logically.

Syntax 1

```
bool operator !=(const TColor& other) const;
```

Description

Returns true if two colors are not equal.

Syntax 2

```
bool operator !=(COLORREF cr) const;
```

Description

Returns true if this color does not match a COLORREF.

TColor::operator COLORREF()

See also TColor

Syntax

```
operator COLORREF() const;
```

Description

Type-conversion operator that returns *Value*.

TColor::Index

See also TColor

Syntax

```
int Index() const;
```

Description

Returns the index value corresponding to this color's *Value* by masking out the two upper bytes. Used when color is a palette index value.

TColor::PalIndex

See also TColor

Syntax

```
TColor PalIndex() const;
```

Description

Returns the palette index corresponding to this color's *Value*. The returned color has the high-order byte set to 1.

TColor::PalRelative

See also TColor

Syntax

```
TColor PalRelative() const;
```

Description

Returns the palette-relative RGB corresponding to this color's *Value*. The returned color has the high-order byte set to 2.

TColor::Red

See also TColor

Syntax

```
uint8 Red() const;
```

Description

Returns the red component of this color's *Value*.

TColor::Rgb

See also TColor

Syntax

```
TColor Rgb() const;
```

Description

Returns the explicit RGB color corresponding to this color's *Value* by masking out the high-order byte.

TColor::SetSysColors

TColor

Syntax

```
static bool SetSysColors(unsigned nelems, const int uiElementIndices[],
  const TColor colors[]);
```

Description

(Presentation Manager only) Sets groups of UI element colors. *nelems* indicates the number of element colors to change and the size of the array parameters, *uiElementIndices* indicates which elements to change, and *colors* indicates what color to change the corresponding element to. Returns true if successful.

TColor::SetValue

TColor

Syntax

```
void SetValue(const COLORREF& value);
```

Description

Changes the color after it has been initialized.

TColor::Value

TColor

Syntax

```
COLORREF Value;
```

Description

The color value of this *TColor* object. *Value* can have three different forms, depending on the application:

- Explicit values for RGB (red, green, blue)
- An index into a logical color palette
- A palette-relative RGB value.

TDropInfo class

Header File
wsyscls.h

Description
TDropInfo is a simple class that supports file-name drag and drop operations using the WM_DROPFILES message. Each *TDropInfo* object has a private handle to the HDROP structure returned by the WM_DOPFILES message.

Public Constructors
- TDropInfo::TDropInfo

Public Member Functions
- DragFinish
- DragQueryFile
- DragQueryFileCount
- DragQueryFileNameLen
- DragQueryPoint
- operator HDROP()

TDropInfo::TDropInfo

TDropInfo

Syntax

```
TDropInfo(HDROP handle);
```

Description

Creates a *TDropInfo* object with *Handle* set to the given *handle*.

TDropInfo::DragFinish

TDropInfo

Syntax

```
void DragFinish();
```

Description

Releases any memory allocated for the transferring of this *TDropInfo* object's files during drag operations.

TDropInfo::DragQueryFile

See also TDropInfo

Syntax

```
uint DragQueryFile(uint index, char far* name, uint nameLen);
```

Description

Retrieves the name of the file and related information for this *i* object. If *index* is set to -1 (0xFFFF), *DragQueryFile* returns the number of dropped files. This is equivalent to calling *DragQueryFileCount*.

If *index* lies between 0 and the total number of dropped files for this object, *DragQueryFile* copies to the *name* buffer (of length *nameLen* bytes) the name of the dropped file that corresponds to *index*, and returns the number of bytes actually copied.

If *name* is 0, *DragQueryFile* returns the required buffer size (in bytes) for the given *index*. This is equivalent to calling *DragQueryFileNameLen*.

TDropInfo::DragQueryFileCount

See also TDropInfo

Syntax

```
uint DragQueryFileCount();
```

Description

Returns the number of dropped files in this *TDropInfo* object. This call is equivalent to calling `DragQueryFile(-1, 0, 0)`.

TDropInfo::DragQueryFileNameLen

See also TDropInfo

Syntax

```
uint DragQueryFileNameLen(uint index);
```

Description

Returns the length of the name of the file in this *TDropInfo* object corresponding to the given index. This call is equivalent to calling `DragQueryFile(index, 0, 0)`.

TDropInfo::DragQueryPoint

See also TDropInfo

Syntax

```
bool DragQueryPoint(TPoint& point);
```

Description

Retrieves the mouse pointer position when this object's files are dropped and copies the coordinates to the given *point* object. *point* refers to the window that received the WM_DROPFILES message. *DragQueryPoint* returns true if the drop occurs inside the window's client area, otherwise false.

TDropInfo::operator HDROP()

TDropInfo

Syntax

```
operator HDROP();
```

Description

Typecasting operator that returns *Handle*.

TFileDroplet class

Header File
wsyscls.h

Description
TFileDroplet encapsulates information about a single dropped file, its name, where it was dropped, and whether or not it was in the client area.

Public Constructors
- TFileDroplet::TFileDroplet

Public Member Functions
- GetInClientArea
- GetName
- GetPoint
- operator ==

TFileDroplet::TFileDroplet

TFileDroplet

Syntax 1

```
TFileDroplet(const char* fileName, TPoint& p, bool inClient);
```

Description

Supports drag and drop.

Syntax 2

```
TFileDroplet(TDropInfo& drop, int i);
```

Description

Constructs a TFileDroplet given a DropInfo and a file index.

The location is relative to the client coordinates, and will have negative values if dropped in the non-client parts of the window.

DragQueryPoint copies that point where the file was dropped and returns whether or not the point is in the client area. Regardless of whether or not the file is dropped in the client or non-client area of the window, you will still receive the file name.

Destructor

```
~TFileDroplet();
```

Description

The destructor for this class.

TFileDroplet::GetInClientArea

TFileDroplet

Syntax

```
bool GetInClientArea() const;
```

Description

Returns **true** if the drop occurred in the client area.

TFileDroplet::GetName

TFileDroplet

Syntax

```
const char* GetName() const;
```

Description

Returns the name of the file dropped.

TFileDroplet::GetPoint

TFileDroplet

Syntax

```
TPoint GetPoint() const;
```

Description

Returns the cursor position at which the file was dropped.

TFileDroplet::operator ==

TFileDroplet

Syntax

```
operator ==(const TFileDroplet& other) const;
```

Description

Returns **true** if the address of this object is equal to the address of the compared object.

TLangId typedef

See also

Header File

lclstrng.h

Syntax

```
typedef unsigned short TLangId;
```

Description

Holds a language ID, a predefined number that represents a base language and dialect. For example, the number 409 represents American English. *TLocaleString* uses the language ID to find the correct translation for strings.

TLocaleString Struct

See also

Header File

lclstrng.h

Description

Designed to provide support for localized registration parameters, the *TLocaleString* Struct defines a localizable substitute for **char*** strings. These strings, which describe both OLE and non-OLE enabled objects to the user, are available in whatever language the user needs. This Struct supports ObjectWindows' Doc/View as well as ObjectComponents' OLE-enabled applications. The public member functions, which supply information about the user's language, the native language, and a description of the string marked for localization, simplify the process of translating and comparing strings in a given language.

To localize the string resource, *TLocaleString* uses several user-entered prefixes to determine what kind of string to translate. Each prefix must be followed by a valid resource identifier (a standard C identifier).The following table lists the prefixes *TLocaleString* uses to localize strings. Each prefix is followed by a sample entry.

Prefix	Description
@ TXY	The string is a series of characters interpreted as a resource ID and is accessed only from a resource file. It is never used directly.
# 1045	The string is a series of digits interpreted as a resource ID and is accessed from a resource file. It is never used directly.
! MyWindow	The string is translated if it is not in the native language; otherwise, this string is used directly.

See the section on localizing symbol names in the ObjectWindows Programmer's Guide for more information about localizing strings.

Public Member Functions

- Compare
- CompareLang
- GetSystemLangId
- GetUserLangId
- IsNativeLangId
- operator =
- operator const char*()
- Translate

Public Data Members

- Module
- NativeLangId
- Null
- Private
- SystemDefaultLangId
- UserDefaultLangId

TLocaleString::Compare

TLocaleString

Syntax

```
int Compare(const char far* str, TLangId lang);
```

Description

Using the specified language (*lang*), *Compare* compares *TLocaleString* with another string. It uses the standard string compare and the language-specific collation scheme. It returns one of the following values.

Return value	Meaning
0	There is no match between the two strings.
1	This string is greater than the other string.
-1	This string is less than the other string.

TLocaleString::CompareLang

TLocaleString

Syntax

```
static int CompareLang(const char far* s1, const char far* s2, TLangId);
```

Description

This function may be re-implemented with enhanced NLS support in another module. Note: That module must be linked in before the library, to override this default implementation.

TLocaleString::GetSystemLangId

See also TLocaleString

Syntax

```
static TLangId GetSystemLangId();
```

Description

Returns the system language ID, which can be the same as the *UserLangId*.

TLocaleString::GetUserLangId

See also TLocaleString

Syntax

```
static TLangId GetUserLangId();
```

Description

Returns the user language ID. For single user systems, this is the same as *LangSysDefault*. The language ID is a predefined number that represents a base language and dialect.

TLocaleString::IsNativeLangId

TLocaleString

Syntax

```
static int IsNativelangId(TLangId lang);
```

Description

Returns **true** if *lang* equals the native system language.

TLocaleString::Module

TLocaleString

Syntax

```
static HINSTANCE Module;
```

Description

The handle of the file containing the resource.

TLocaleString::NativeLangId

TLocaleString

Syntax

```
static TLangId NativeLangId;
```

Description

The base language ID of non-localized strings.

TLocaleString::Null

TLocaleString

Syntax

```
static TLocaleString Null;
```

Description

A null string.

TLocaleString::operator const char*

TLocaleString

Syntax

```
operator const char* ();
```

Description

Returns the current character string in the translation.

TLocaleString::operator =

TLocaleString

Syntax

```
void operator = (const char* str);
```

Description

Assigns the string (*str*) to this locale string.

TLocaleString::Private

TLocaleString

Syntax

```
const char* Private;
```

Description

A string pointer used internally.

TLocaleString::SystemDefaultLangId

TLocaleString

Syntax

```
static TLangId SystemDefaultLangId;
```

Description

The default language identifier.

TLocaleString::Translate

TLocaleString

Syntax

```
const char* Translate(TLangId lang);
```

Description

Translates the string to the given language. *Translate* follows this order of preference in order to choose a language for translation:

1. Base language and dialect.
2. Base language and no dialect.
3. Base language and another dialect.
4. The native language of the resource itself.
5. Returns 0 if unable to translate the string. (This can happen only if an @ or # prefix is used; otherwise, the ! prefix indicates that the string following is the native language itself.)

TLocaleString::UserDefaultLangId

TLocaleString

Syntax

```
static TLangId UserDefaultLangId;
```

Description

The user-defined default language identifier.

TMsgThread class

Header File
msgthred.h

Description
TMsgThread implements basic behavior for threads that own message queues, including mutex locking for the queue. This class provides message queue oriented thread class support. TMsgThread degenerates to a simple message queue owner under non-threaded environments.

Type Definitions
- TCurrent

Public Constructors
- TMsgThread::TMsgThread

Public Data Members
- BreakMessageLoop
- LoopRunning
- MessageLoopResult

Public Member Functions
- EnableMultiThreading
- FlushQueue
- GetMutex
- IdleAction
- IsRunning
- MessageLoop
- ProcessMsg
- PumpWaitingMessages

Protected Constructors
- TMsgThread

Protected Member Functions
- InitInstance
- Mutex
- Run
- TermInstance

TMsgThread::TCurrent enum

TMsgThread

Syntax

```
enum TCurrent {Current};
```

TMsgThread::TMsgThread

TMsgThread

Syntax

```
TMsgThread(TCurrent);
```

Description

Attaches to the current running thread. This is often the initial process thread, or even the only thread for non-threaded systems.

TMsgThread::BreakMessageLoop

TMsgThread

Syntax

```
bool BreakMessageLoop;
```

Description

The message loop is broken via WM_QUIT.

TMsgThread::LoopRunning

TMsgThread

Syntax

```
bool LoopRunning;
```

Description

Tracks whether the loop is running.

TMsgThread::MessageLoopResult

TMsgThread

Syntax

```
int MessageLoopResult;
```

Description

Returns the value from the message loop.

TMsgThread::EnableMultiThreading

TMsgThread

Syntax

```
void EnableMultiThreading(bool enable);
```

Description

Enables or disables the use of the mutex to synchronize access to the message queue. TMsgThread will only lock its message queue mutex when enabled. Real multi-threading requires compiler and RTL support.

TMsgThread::FlushQueue

TMsgThread

Syntax

```
void FlushQueue();
```

Description

Flushes all real messages from the message queue.

TMsgThread::GetMutex

TMsgThread

Syntax

```
TMutex* GetMutex();
```

Description

Gets this message thread's mutex. Returns 0 if mutexes are not supported, or are not enabled for this thread.

TMsgThread::IdleAction

TMsgThread

Syntax

```
virtual bool IdleAction(long idleCount);
```

Description

Called each time there are no messages in the queue. Idle count is incremented each time, and zeroed when messages are pumped. Returns whether or not more processing needs to be done.

TMsgThread::IsRunning

TMsgThread

Syntax

```
bool IsRunning() const;
```

Description

Returns true if this queue thread is running its message loop.

TMsgThread::MessageLoop

TMsgThread

Syntax

```
virtual int MessageLoop();
```

Description

Retrieves and processes messages from the thread's message queue using PumpWaitingMessages() until BreakMessageLoop becomes true. Catches exceptions to post a quit message and cleanup before resuming.

TMsgThread::ProcessMsg

TMsgThread

Syntax

```
virtual bool ProcessMsg(MSG& msg);
```

Description

Called for each message that is pulled from the queue, to perform all translation and dispatching.

Returns true to drop out of the pump.

TMsgThread::PumpWaitingMessages

TMsgThread

Syntax

```
bool PumpWaitingMessages();
```

Description

The inner message loop. Retrieves and processes messages from the OWL application's message queue until it is empty. Set BreakMessageLoop if a WM_QUIT passes through.

Call ProcessAppMsg() for each message to allow special pre-handling of the message.

TMsgThread::TMsgThread (Protected)

TMsgThread

Syntax

```
TMsgThread();
```

TMsgThread::InitInstance

TMsgThread

Syntax

```
virtual void InitInstance();
```

Description

Handles initialization for each executing instance of the message thread. Derived classes can override this to perform initialization for each instance.

TMsgThread::Mutex

TMsgThread

Syntax

```
TAppMutex Mutex;
```

Description

Prevents multiple threads from processing messages at the same time.

TMsgThread::Run

TMsgThread

Syntax

```
virtual int Run();
```

Description

Runs this message thread, returns when the message queue quits Initialize instances. Runs the thread's message loop. Each of the virtual functions called are expected to throw an exception if there is an error.

Exceptions that are not handled, that is, where the status remains non-zero, are propagated out of this function. The message queue is still flushed and TermInstance called.

TMsgThread::TermInstance

TMsgThread

Syntax

```
virtual int TermInstance(int status);
```

Description

Handles termination for each executing instance of the message thread. Called at the end of a Run() with the final return status.

TPaletteEntry class

Header File
color.h

Description
TPaletteEntry is a support class derived from the structure *tagPALETTEENTRY*. The latter is defined as follows:

```
typedef struct tagPALETTEENTRY {
   uint8  peRed;
   uint8  peGreen;
   uint8  peBlue;
   uint8  peFlags;
} PALETTEENTRY;
```

The members *peRed*, *peGreen*, and *peBlue* specify the red, green, and blue intensity-values for a palette entry.

The *peFlags* member can be set to NULL or one of the following values:

Value	Meaning
PC_EXPLICIT	Specifies that the low-order word of the logical palette entry designates a hardware palette index. This flag allows the application to show the contents of the display device palette.
PC_NOCOLLAPSE	Specifies that the color be placed in an unused entry in the system palette instead of being matched to an existing color in the system palette. If there are no unused entries in the system palette, the color is matched normally. Once this color is in the system palette, colors in other logical palettes can be matched to this color.
PC_RESERVED	Specifies that the logical palette entry be used for palette animation; this prevents other windows from matching colors to this palette entry since the color frequently changes. If an unused system-palette entry is available, this color is placed in that entry. Otherwise, the color is available for animation.

TPaletteEntry is used in conjunction with the classes TPalette and TColor to simplify logical color-palette operations. Constructors are provided to create *TPaletteEntry* objects from explicit COLORREF and RGB values, or from *TColor* objects.

Public Constructors
TPaletteEntry::TPaletteEntry

Public Member Functions
operator ==

TPaletteEntry::TPaletteEntry

TPaletteEntry

Syntax 1

```
TPaletteEntry(int r, int g, int b, int f = 0);
```

Description

Creates a palette entry object with *peRed*, *peGreen*, *peBlue*, and *peFlags* set to *r*, *g*, *b*, and *f*, respectively.

Syntax 2

```
TPaletteEntry(TColor c);
```

Description

Creates a palette entry object with *peRed*, *peGreen*, *peBlue*, and *peFlags* set to *r*, *g*, *b*, and *f*, respectively.

TPaletteEntry::operator ==

See also TPaletteEntry

Syntax

```
bool operator ==(COLORREF cr) const;
```

Description

Returns true if the palette entries have the same color components.

TPoint class

Header File
geometry.h

Description
TPoint is a support class, derived from *tagPOINT*. The *tagPOINT* struct is defined as

```
struct tagPOINT {
   int x;
   int y;
};
```

TPoint encapsulates the notion of a two-dimensional point that usually represents a screen position. *TPoint* inherits two data members, the coordinates *x* and *y*, from *tagPOINT*. Member functions and operators are provided for comparing, assigning, and manipulating points. Overloaded **<<** and **>>** operators allow chained insertion and extraction of **TPoint** objects with streams.

Public Constructors
- TPoint::TPoint

Public Member Functions
- Magnitude
- Offset
- OffsetBy
- operator +
- operator -
- operator ==
- operator +=
- operator -=
- operator !=
- operator >>
- operator <<
- X
- Y

TPoint::TPoint

See also TPoint

Syntax 1

```
TPoint();
```

Description

The default *TPoint* constructor.

Syntax 2

```
TPoint(int _x, int _y);
```

Description

Creates a *TPoint* object with the given coordinates.

Syntax 3

```
TPoint(const POINT& point);
```

Description

Creates a *TPoint* object with *x* = *point.x*, *y* = *point.y*.

Syntax 4

```
TPoint(const SIZE& size);
```

Description

Creates a *TPoint* object with *x* = *size.cx* and *y* = *size.cy*.

Syntax 5

```
TPoint(uint32 dw);
```

Description

Creates a *TPoint* object with *x* = LOWORD(*dw*), *y* = HIWORD(*dw*)).

TPoint::Magnitude

TPoint

Syntax

```
int Magnitude() const;
```

Description

Returns the distance between the origin and the point.

TPoint::Offset

See also TPoint

Syntax

```
TPoint& Offset(int dx, int dy);
```

Description

Offsets this point by the given delta arguments. This point is changed to (*x* + *dx*, *y* + *dy*). Returns a reference to this point.

TPoint::OffsetBy

See also TPoint

Syntax

```
TPoint OffsetBy(int dx, int dy) const;
```

Description

Calculates an offset to this point using the given displacement arguments. Returns the point (*x* + *dx*, *y* + *dy*). This point is not changed.

TPoint::operator +

See also TPoint

Syntax

```
TPoint operator +(const TSize& size) const;
```

Description

Calculates an offset to this point using the given size argument as the displacement. Returns the point (*x* + *size.cx*, *y* + *size.cy*). This point is not changed.

TPoint::operator -

See also TPoint

Syntax 1

```
TPoint operator -(const TSize& size) const;
```

Description

Calculates a negative offset to this point using the given *size* argument as the displacement. Returns the point (*x* - *size.cx*, *y* - *size.cy*). This point is not changed.

Syntax 2

```
TSize operator -(const TPoint& point) const;
```

Description

Calculates a distance from this point to the *point* argument. Returns the *TSize* object (*x* - *point.x*, *y* - *point.y*). This point is not changed.

Syntax 3

```
TPoint operator -() const;
```

Description

Returns the point (-*x*, -*y*). This point is not changed.

TPoint::operator ==

See also TPoint

Syntax

```
bool operator ==(const TPoint& other) const;
```

Description

Returns true if this point is equal to the *other* point; otherwise returns false.

TPoint::operator +=

See also TPoint

Syntax

```
TPoint& operator +=(const TSize& size);
```

Description

Offsets this point by the given *size* argument. This point is changed to(*x* + *size.cx*, *y* + *size.cy*). Returns a reference to this point.

TPoint::operator -=

See also TPoint

Syntax

```
TPoint& operator -=(const TSize& size);
```

Description

Negatively offsets this point by the given *size* argument. This point is changed to (*x* - *size.cx*, *y* - *size.cy*). Returns a reference to this point.

TPoint::operator !=

See also TPoint

Syntax

```
bool operator !=(const TPoint& other) const;
```

Description

Returns false if this point is equal to the *other* point; otherwise returns true.

TPoint::operator >>

See also TPoint

Syntax 1

```
ipstream& operator >>(ipstream& is, TPoint& p);
```

Description

Extracts a *TPoint* object from persistent stream *is*, and copies it to *p*. Returns a reference to the resulting stream, allowing the usual chaining of **<<** operations.

Syntax 2

```
istream& operator >>(istream& is, TPoint& p);
```

Description

Extracts a *TPoint* object from stream *is*, and copies it to *p*. Returns a reference to the resulting stream, allowing the usual chaining of **>>** operations.

TPoint::operator <<

See also TPoint

Syntax 1

```
opstream& operator <<(opstream& os, const TPoint& p);
```

Description

Inserts the given *TPoint* object *p* into persistent stream *os*. Returns a reference to the resulting stream, allowing the usual chaining of **>>** operations.

Syntax 2

```
ostream& operator <<(ostream& os, const TPoint& p);
```

Description

Formats and inserts the given *TPoint* object *p* into the *ostream os*. The format is "(x,y)". Returns a reference to the resulting stream, allowing the usual chaining of **<<** operations.

TPoint::X

TPoint

Syntax

```
int X() const;
```

Description

Returns the x coordinate of the point.

TPoint::Y

TPoint

Syntax

```
int Y() const;
```

Description

Returns the y coordinate of the point.

TPointF class

Header File
geometry.h

Description
TPointF is similar to TPoint, but uses floating variables rather than integers.

Public Constructors
- TPointF::TPointF

Public Member Functions
- Offset
- OffsetBy
- operator !=
- operator +
- operator +=
- operator -
- operator -
- operator -=
- operator ==
- X
- Y

TPointF::TPointF

TPointF

Syntax 1

```
TPointF();
```

Description

Default constructor that does nothing.

Syntax 2

```
TPointF(float _x, float _y);
```

Description

Constructor that initializes the location.

Syntax 3

```
TPointF(const TPointF far& point);
```

Description

Constructor that copies the location.

TPointF::Offset

TPointF

Syntax

```
TPointF& Offset(float dx, float dy);
```

Description

Moves the point by an offset.

TPointF::OffsetBy

TPointF

Syntax

```
TPointF OffsetBy(float dx, float dy) const;
```

Description

Creates a new point that is offset from the current point.

TPointF::operator !=

TPointF

Syntax

```
bool operator !=(const TPointF& other) const;
```

Description

Returns true if the points are not at the same location.

TPointF::operator +

TPointF

Syntax

```
TPointF operator +(const TPointF& size) const;
```

Description

Returns a new point (x+cx, y+cy).

TPointF::operator +=

TPointF

Syntax

```
TPointF& operator +=(const TPointF& size);
```

Description

Returns the new point moved by the offset.

TPointF::operator -

TPointF

Form 1

```
TPointF operator -() const;
```

Form 2

```
TPointF operator -(const TPointF& point) const;
```

Description

Form 1: Returns the new point subtracted from the current.

Form 2: Returns the negative of the point.

TPointF::operator -=

TPointF

Syntax

```
TPointF& operator -=(const TPointF& size);
```

Description

Returns the new point subtracted from the current.

TPointF::operator ==

TPointF

Syntax

```
bool operator ==(const TPointF& other) const;
```

Description

Returns true if the points are at the same location.

TPointF::X

TPointF

Syntax

```
float X() const;
```

Description

Returns the X component of the point.

TPointF::Y

TPointF

Syntax

```
float Y() const;
```

Description

Returns the Y component of the point.

TPointL class

Header File
geometry.h

Description
TPointL is similar to TPoint, but uses long rather than int variables.

Public Constructors
- TPointL::TPointL

Public Member Functions
- Offset
- OffsetBy
- operator !=
- operator +
- operator +=
- operator -
- operator -=
- operator ==
- X
- Y

TPointL::TPointL

TPointL

Syntax 1

```
TPointL();
```

Description

Default constructor that does nothing.

Syntax 2

```
TPointL(long _x, long _y);
```

Description

Constructs the point to a specific location.

Syntax 3

```
TPointL(const POINTL far& point);
```

Description

Alias constructor that initializes from an existing point.

Syntax 4

```
TPointL(const TPointL far& point);
```

Description

Makes a copy of the location.

TPointL::Offset

TPointL

Syntax

```
TPointL& Offset(long dx, long dy);
```

Description

Returns the point (x+dx, y+dy), shifting the point by the offset.

TPointL::OffsetBy

TPointL

Syntax

```
TPointL OffsetBy(long dx, long dy) const;
```

Description

Returns the new point (x+dx, y+dy). Creates a new point shifted by the offset, preserving the original point.

TPointL::operator !=

TPointL

Syntax

```
bool operator !=(const TPointL& other) const;
```

Description

Returns true if the positions are not the same.

TPointL::operator +

TPointL

Syntax

```
TPointL operator +(const TSize& size) const;
```

Description

Returns the new point (x+cx, y+cy).

TPointL::operator +=

TPointL

Syntax

```
TPointL& operator +=(const TSize& size);
```

Description

Returns the point (x+cx, y+cy).

TPointL::operator -

TPointL

Syntax 1

```
TPointL operator -() const;
```

Description

Returns the negative of the point.

Syntax 2

```
TPointL operator -(const TSize& size) const;
```

Description

Returns the new point (x-cx, y-cy).

Syntax 3

```
TPointL operator -(const TPointL& point) const;
```

Description

Returns the difference between the two points.

TPointL::operator -=

TPointL

Syntax

```
TPointL& operator -=(const TSize& size);
```

Description

Returns the point (x-cx, y-cy).

TPointL::operator ==

TPointL

Syntax

```
bool operator ==(const TPointL& other) const;
```

Description

Returns true if positions are the same.

TPointL::X

TPointL

Syntax

```
long X() const;
```

Description

Returns the X component of the point.

TPointL::Y

TPointL

Syntax

```
long Y() const;
```

Description

Returns the Y component of the point.

TProcInstance class

Header File

wsyscls.h

Description

A ProcInstance object. This encapsulates the MakeProcInstance call, which is really only needed in old Win3.X real mode. This exists now for Owl 2.x compatibility only.

Designed for Win16 applications, *TProcInstance* handles creating and freeing an instance thunk, a piece of code created for use with exported callback functions. (A callback function is a function that exists within a program but is called from outside the program by a Windows library routine, for example, a dialog box function.)

For Win32 applications, *TProcInstance* is non-functional. The address returned from *TProcInstance* can be passed as a parameter to callback functions, window subclassing functions, or Windows dialog box functions.

See the Windows API online Help for more information about *MakeProcInstance*, which creates an instance thunk for the function and *FreeProcInstance*, which frees an instance thunk. For more information about exporting callback functions, see the Borland C++ Programmer's Guide.

Public Constructors

TProcInstance::TProcInstance

Public Member Functions

FARPROC

TProcInstance::TProcInstance

See also

Syntax

```
TProcInstance(FARPROC p);
```

Description

Makes a *TProcInstance*, passing *p* as the address of the procedure. Under Win16, calls *::MakeProcInstance* to make an instance thunk for *p*. Under Win32, the constructor just saves *p*.

Destructor

```
~TProcInstance()
```

Description

Under WIN16, frees the instance thunk.

TProcInstance::operator FARPROC

Syntax

```
operator FARPROC();
```

Description

Under WIN16, returns the instance thunk. Under Win32, returns *p* from the constructor.

TProfile class

Header File
profile.h

Description
An instance of *TProfile* encapsulates a setting within a system file, often referred to as a *profile* or *initialization* file. Examples of this type of file include the Windows initialization files SYSTEM.INI and WIN.INI. Within the system file itself, the individual settings are grouped within sections. For example,

```
[Diagnostics]     ; section name
Enabled=0         ; setting
```

For a setting, the value to the left of the equal sign is called the *key.* The value to the right of the equal sign, the *value*, can be either an integer or a string data type.

Public Constructors
- TProfile::TProfile

Public Member Functions
- Flush
- GetInt
- GetString
- WriteInt
- WriteString

TProfile::TProfile

TProfile

Constructor Syntax

```
TProfile(const char* section, const char* filename=0);
```

Description

Constructs a *TProfile* object for the indicated *section* within the profile file specified by *filename*. If the file name is not provided, the file defaults to the system profile file; for example, WIN.INI under Windows .

Destructor Syntax

```
~TProfile();
```

Description

Destroys the *TProfile* object.

TProfile::Flush

TProfile

Syntax

```
void Flush();
```

Description

Makes sure that all written profile values are flushed to the actual file.

TProfile::GetInt

TProfile

Syntax

```
int GetInt(const char* key, int defaultInt = 0);
```

Description

Looks up and returns the integer value associated with the given string, *key*. If *key* is not found, the default value, *defaultInt*, is returned.

TProfile::GetString

TProfile

Syntax

```
bool GetString(const char* key, char buff[], unsigned buffSize, const char*
  defaultString = "");
```

Description

Looks up and returns the string value associated with the given *key* string. The string value is copied into *buff*, up to *buffSize* bytes. If the key is not found, *defaultString* provides the default value. If a 0 key is passed, all section values are returned in *buff*.

TProfile::WriteInt

TProfile

Syntax

```
bool WriteInt(const char* key, const char* int value);
```

Description

Looks up the key and replaces its value with the integer value passed (*int*). If the key is not found, *WriteInt* makes a new entry. Returns **true** if successful.

TProfile::WriteString

TProfile

Syntax

```
bool WriteString(const char* key, const char* str);
```

Description

Looks up the key and replaces its value with the string value passed (*str*). If the key is not found, *WriteString* makes a new entry. Returns **true** if successful.

TRect Class

Header File
geometry.h

Description
TRect is a mathematical class derived from *tagRect*. The *tagRect* struct is defined as

```
struct tagRECT {
   int left;
   int top;
   int right;
   int bottom;
};
```

TRect encapsulates the properties of rectangles with sides parallel to the x- and y-axes. In ObjectWindows, these rectangles define the boundaries of windows, boxes, and clipping regions. *TRect* inherits four data members from *tagRect left*, *top*, *right*, and *bottom*. These represent the top left and bottom right (*x*, *y*) coordinates of the rectangle. Note that *x* increases from left to right, and *y* increases from top to bottom.

TRect places no restrictions on the relative positions of top left and bottom right, so it is legal to have *left* > *right* and *top* > *bottom*. However, many manipulations--such as determining width and height, and forming unions and intersections--are simplified by normalizing the *TRect* objects involved. Normalizing a rectangle means interchanging the corner point coordinate values so that *left* < *right* and *top* < *bottom*. Normalization does not alter the physical properties of a rectangle. *myRect*.*Normalized* creates normalized copy of *myRect* without changing *myRect*, while *myRect*.*Normalize* changes *myRect* to a normalized format. Both members return the normalized rectangle.

TRect constructors are provided to create rectangles from either four **int**s, two TPoint objects, or one *TPoint* and one TSize object. In the latter case, the *TPoint* object specifies the top left point (also known as the rectangle's origin) and the *TSize* object supplies the width and height of the rectangle. Member functions perform a variety of rectangle tests and manipulations. Overloaded **<<** and **>>** operators allow chained insertion and extraction of *TRect* objects with streams.

Public Constructors
- TRect::TRect

Public Member Functions
- Area
- Bottom
- BottomLeft
- BottomRight
- Contains
- Height
- Inflate
- InflatedBy
- IsEmpty
- IsNull
- Left
- MovedTo
- MoveTo
- Normalize
- Normalized
- Offset

OffsetBy
operator+
operator -
operator &
operator |
operator ==
operator !=
operator +=
operator -=
operator &=
operator |=
Right
Set
SetEmpty
SetNull
SetWH
Size
Subtract
Top
TopLeft
TopRight
Touches
TPoint
Width
>>
<<
X
Y

TRect::TRect

See also TRect

Constructors

Syntax 1

```
TRect();
```

Description

The default constructor.

Syntax 2

```
TRect(const tagRECT far& rect);
```

Description

Copies from an existing rectangle.

Syntax 3

```
TRect(const TRect far& rect);
```

Description

Copies the given *rect* to this object.

Syntax 4

```
TRect(int _left, int _top, int _right, int _bottom);
```

Description

Creates a rectangle with the given values.

Syntax 5

```
TRect(const TPoint& upLeft, const TPoint& loRight);
```

Description

Creates a rectangle with the given top left and bottom right points.

Syntax 6

```
TRect(const TPoint& origin, const TSize& extent);
```

Description

Creates a rectangle with its origin (top left) at *origin*, width at *extent.cx*, height at *extent.cy*.

TRect::Area

See also TRect

Syntax

```
long Area() const;
```

Description

Returns the area of this rectangle.

TRect::Bottom

TRect

Syntax

```
int Bottom() const;
```

Description

Returns the bottom value.

TRect::BottomLeft

See also TRect

Syntax

```
TPoint BottomLeft() const;
```

Description

Returns the *TPoint* object representing the bottom left corner of this rectangle.

TRect::BottomRight

See also TRect

Syntax 1

```
const TPoint& BottomRight() const;
```

Syntax 2

```
TPoint& BottomRight();
```

Description

Returns the *TPoint* object representing the bottom right corner of this rectangle.

TRect::Contains

See also TRect

Syntax 1

```
bool Contains(const TPoint& point) const;
```

Description

Returns **true** if the given *point* lies within this rectangle; otherwise, it returns **false**. If *point* is on the left vertical or on the top horizontal borders of the rectangle, *Contains* also returns **true**, but if *point* is on the right vertical or bottom horizontal borders, *Contains* returns **false**.

Syntax 2

```
bool Contains(const TRect& other) const;
```

Description

Returns **true** if the other rectangle lies on or within this rectangle; otherwise, it returns **false**.

TRect::Height

See also TRect

Syntax

```
int Height() const;
```

Description

Returns the height of this rectangle (*bottom* - *top*).

TRect::Inflate

See also TRect

Syntax 1

```
TRect& Inflate(int dx, int dy);
```

Syntax 2

```
TRect& Inflate(const TSize& delta);
```

Description

Inflates a rectangle inflated by the given delta arguments. In the first version, the top left corner of the returned rectangle is (*left* - *dx*, *top* - *dy*), while its bottom right corner is (*right* + *dx*, *bottom* + *dy*). In the second version the new corners are (*left* - *size.cx*, *top* - *size.cy*) and (*right* + *size.cx*, *bottom* + *size.cy*).

TRect::InflatedBy

See also TRect

Syntax 1

```
TRect InflatedBy(int dx, int dy) const;
```

Syntax 2

```
TRect InflatedBy(const TSize& size) const;
```

Description

Returns a rectangle inflated by the given delta arguments. In the first version, the top left corner of the returned rectangle is (*left* - *dx*, *top* - *dy*), while its bottom right corner is (*right* + *dx*, *bottom* + *dy*). In the second version the new corners are (*left* - *size.cx*, *top* - *size.cy*) and (*right* + *size.cx*, *bottom* + *size.cy*). The calling rectangle object is unchanged.

TRect::IsEmpty

See also TRect

Syntax

```
bool IsEmpty() const;
```

Description

Returns **true** if *left* >= *right* or *top* >= *bottom*; otherwise, returns **false**.

TRect::IsNull

See also TRect

Syntax

```
bool IsNull() const;
```

Description

Returns **true** if left, *right*, *top*, and *bottom* are all 0; otherwise, returns **false**.

TRect::Left

TRect

Syntax

```
int Left() const;
```

Description

Returns the left value.

TRect::MovedTo

TRect

Syntax

```
TRect MovedTo(int x, int y);
```

Description

Moves the upper left point of the rectangle while maintaining the current dimensions.

TRect::MoveTo

TRect

Syntax

```
TRect& MoveTo(int x, int y);
```

Description

Move the upper left corner of the rectangle to a new location and maintain the current dimensions.

TRect::Normalize

See also TRect

Syntax

```
TRect& Normalize();
```

Description

Normalizes this rectangle by switching the *left* and *right* data member values if *left* > *right*, and switching the *top* and *bottom* data member values if *top* > *bottom*. *Normalize* returns the normalized rectangle. A valid but nonnormal rectangle might have *left* > *right* or *top* > *bottom* or both. In such cases, many manipulations (such as determining width and height) become unnecessarily complicated. Normalizing a rectangle means interchanging the corner point values so that *left* < *right* and *top* < *bottom*. The physical properties of a rectangle are unchanged by this process.

TRect::Normalized

See also TRect

Syntax

```
TRect Normalized() const;
```

Description

Returns a normalized rectangle with the top left corner at (*Min*(*left*, *right*), *Min*(*top*, *bottom*)) and the bottom right corner at (*Max*(*left*, *right*), *Max*(*top*, *bottom*)). The calling rectangle object is unchanged. A valid but nonnormal rectangle might have *left* > *right* or *top* > *bottom* or both. In such cases, many manipulations (such as determining width and height) become unnecessarily complicated. Normalizing a rectangle means interchanging the corner point values so that *left* < *right* and *top* < *bottom*. The physical properties of a rectangle are unchanged by this process.

Note that many calculations assume a normalized rectangle. Some Windows API functions behave erratically if an inside-out *Rect* is passed.

TRect::Offset

See also TRect

Syntax

```
TRect& Offset(int dx, int dy);
```

Description

Changes this rectangle so its corners are offset by the given delta values. The revised rectangle has a top left corner at (*left* + *dx*, *top* + *dy*) and a right bottom corner at (*right* + *dx*, *bottom* + *dy*). The revised rectangle is returned.

TRect::OffsetBy

See also TRect

Syntax

```
TRect OffsetBy(int dx, int dy) const;
```

Description

Returns a rectangle with the corners offset by the given delta values. The returned rectangle has a top left corner at (*left* + *dx*, *top* + *dy*) and a right bottom corner at (*right* + *dx*, *bottom* + *dy*).

TRect::operator +

See also TRect

Syntax

```
TRect operator +(const TSize& size) const;
```

Description

Returns a rectangle offset positively by the delta values' given sizes. The returned rectangle has a top left corner at (*left* + *size.x*, *top* + *size.y*) and a right bottom corner at (*right* + *size.x*, *bottom* + *size.y*). The calling rectangle object is unchanged.

TRect::operator -

See also TRect

Syntax

```
TRect operator -(const TSize& size) const;
```

Description

Returns a rectangle offset negatively by the delta values' given sizes. The returned rectangle has a top left corner at (*left* - *size.cx*, *top* - *size.cy*) and a right bottom corner at (*right* - *size.cx*, *bottom* - *size.cy*). The calling rectangle object is unchanged.

TRect::operator &

See also TRect

Syntax

```
TRect operator &(const TRect& other) const;
```

Description

Returns the intersection of this rectangle and the *other* rectangle. The calling rectangle object is unchanged. Returns a NULL rectangle if the two don't intersect.

TRect::operator |

See also TRect

Syntax

```
TRect operator |(const TRect& other) const;
```

Description

Returns the union of this rectangle and the *other* rectangle. The calling rectangle object is unchanged.

TRect::operator ==

See also TRect

Syntax

```
bool operator ==(const TRect& other) const;
```

Description

Returns **true** if this rectangle has identical corner coordinates to the *other* rectangle; otherwise, returns **false**.

TRect::operator !=

See also TRect

Syntax

```
bool operator !=(const TRect& other) const;
```

Description

Returns **false** if this rectangle has identical corner coordinates to the *other* rectangle; otherwise, returns **true**.

TRect::operator +=

See also TRect

Syntax

```
TRect& operator +=(const TSize& delta);
```

Description

Changes this rectangle so its corners are offset by the given delta values, *delta.x* and *delta.y*. The revised rectangle has a top left corner at (*left* + *delta.x*, *top* + *delta.y*) and a right bottom corner at (*right* + *delta.x*, *bottom* + *delta.y*). The revised rectangle is returned.

TRect::operator -=

See also TRect

Syntax

```
TRect& operator -=(const TSize& delta);
```

Description

Changes this rectangle so its corners are offset negatively by the given delta values, *delta.x* and *delta.y*. The revised rectangle has a top left corner at (*left* - *delta.x*, *top* - *delta.y*) and a right bottom corner at (*right* - *delta.x*, *bottom* - *delta.y*). The revised rectangle is returned.

TRect::operator &=

See also TRect

Syntax

```
TRect& operator &=(const TRect& other);
```

Description

Changes this rectangle to its intersection with the *other* rectangle. This rectangle object is returned. Returns a NULL rectangle if there is no intersection.

TRect::operator |=

See also TRect

Syntax

```
TRect& operator |=(const TRect& other);
```

Description

Changes this rectangle to its union with the *other* rectangle. This rectangle object is returned.

TRect::operator TPoint*()

See also TRect

Form 1

```
operator const TPoint*() const;
```

Form 2

```
operator TPoint*()
```

Description

Type conversion operators converting the pointer to this rectangle to type pointer to *TPoint*.

TRect::Right

TRect

Syntax

```
int Right() const;
```

Description

Returns the right value.

TRect::Set

TRect

Syntax

```
void Set(int _left, int _top, int _right, int _bottom);
```

Description

Repositions and resizes this rectangle to the given values.

TRect::SetEmpty

TRect

Syntax

```
void SetEmpty();
```

Description

Empties this rectangle by setting *left*, *top*, *right*, and *bottom* to 0.

TRect::SetNull

TRect

Syntax

```
void SetNull();
```

Description

Sets the *left*, *top*, *right*, and *bottom* of the rectangle to 0.

TRect::SetWH

TRect

Syntax

```
void SetWH(int _left, int _top, int w, int h);
```

Description

Determines the rectangle, given its upperleft point, width, and height.

TRect::Size

See also TRect

Syntax

```
TSize Size() const;
```

Description

Returns a *TSize* object representing the width and height of this rectangle.

TRect::Subtract

TRect

Syntax

```
int Subtract(const TRect& other, TRect result[]) const;
```

Description

Determines the parts of this rect that do not lie within "other" region. The resulting rectangles are placed in the "result" array.

Returns the resulting number of rectangles. This number will be 1, 2, 3, or 4.

TRect::Top

TRect

Syntax

```
int Top() const;
```

Description

Returns the top value.

TRect::TopLeft

See also TRect

Syntax 1

```
const TPoint& TopLeft() const;
```

Syntax 2

```
TPoint& TopLeft();
```

Description

Returns the *TPoint* object representing the top left corner of this rectangle.

TRect::TopRight

See also TRect

Syntax

```
TPoint TopRight() const;
```

Description

Returns the TPoint object representing the top right corner of this rectangle.

TRect::Touches

See also TRect

Syntax

```
bool Touches(const TRect& other) const;
```

Description

Returns **true** if the *other* rectangle shares any interior points with this rectangle; otherwise, returns **false**.

TRect::Width

See also TRect

Syntax

```
int Width() const;
```

Description

Returns the width of this rectangle (*right* - *left*).

TRect::X

TRect

Syntax

```
int X() const;
```

Description

Returns the left value.

TRect::Y

TRect

Syntax

```
int Y() const;
```

Description

Returns the top value.

TRect::operator >>

See also TRect

Syntax

```
ipstream& _BIDSFUNC operator >>(ipstream& is, TRect& r);
```

Description

Extracts a *TRect* object from *is*, the given input stream, and copies it to *r*. Returns a reference to the resulting stream, allowing the usual chaining of **>>** operations.

TRect::operator <<

See also TRect

Syntax 1

```
opstream& _BIDSFUNC operator <<(opstream& os, const TRect& r);
```

Description

Inserts the given *TRect* object, *r*, into the *opstream*, *os*. Returns a reference to the resulting stream, allowing the usual chaining of **<<** operations.

Syntax 2

```
ostream& _BIDSFUNC operator <<(ostream& os, const TRect& r);
```

Description

Formats and inserts the given *TRect* object, *r*, into the *ostream*, *os.* The format is (*r.left*, *r.top*)(*r.right*, *r.bottom*). Returns a reference to the resulting stream and allows the usual chaining of **<<** operations.

TRegistry class

Header File
registry.h

Description
TRegistry provides high level stream and list access to the registry.

Public Member Functions
- Unregister
- Update
- Validate

TRegistry::Unregister

TRegistry

Syntax

```
static int Unregister(TRegList& regInfo, TUnregParams* params, TRegItem*
  overrides = 0);
```

Description

Unregisters entries given a reglist. An optional overrides regItem. Returns the number of errors from deleting keys.

TRegistry::Update

TRegistry

Syntax

```
static void Update(TRegKey& baseKey, istream& in);
```

Description

Walks through an input stream and uses `basekey\key\key=data` lines to set registry entries.

Has named value support in the form: `basekey\key\key|valuename=data`

TRegistry::Validate

TRegistry

Syntax

```
static int Validate(TRegKey& baseKey, istream& in);
```

Description

Walks through an input stream and uses `basekey\key\key=data` lines to check registry entries.

Returns the number of differences. Zero means a complete match.

Has named value support in the following form: `basekey\key\key|valuename=data`

TRegItem Struct

See also

Header File

locale.h

Description

TRegItem defines localizable values for parameters or subkeys. These values can be passed to TLocaleString, which defines a localizable substitute for **char***. *TRegItem* contains several undocumented functions that are used privately by the registration macros REGFORMAT and REGSTATUS.

Public Data Members

Key

Value

Public Member Functions

OverflowCheck

RegFlags

RegFormat

RegVerbOpt

TRegItem::Key

See also TRegItem

Syntax

```
char* Key;
```

Description

Contains a non-localizable item name, such as *clsid* or *progid*.

TRegItem::Value

See also TRegItem

Syntax

```
TLocaleString Value;
```

Description

Contains a localizable value (for example, "My Sample App") associated with an item name.

TRegItem::OverflowCheck

TRegItem

Syntax

```
static void OverflowCheck();
```

TRegItem::RegFlags

TRegItem

Syntax

```
static char* RegFlags(long flags, TRegFormatHeap& heap);
```

TRegItem::RegFormat

TRegItem

Syntax 1

```
static char* RegFormat(const char* f, int a, int t, int d, TRegFormatHeap&
  heap);
```

Description

Registers data formats for the object.

Syntax 2

```
static char* RegFormat(int f, int a, int t, int d, TRegFormatHeap& heap);
```

Description

Registers data formats for the object.

TRegItem::RegVerbOpt

TRegItem

Syntax

```
static char* RegVerbOpt(int mf, int sf, TRegFormatHeap& heap);
```

Description

Registers the verb option.

TRegKey class

Header File
registry.h

Description
TRegKey is the encapsulation of a registration key.

Public Constructors
- TRegKey::TRegKey

Public Member Functions
- ClassesRoot
- ClassesRootClsid
- CurrentConfig
- CurrentUser
- DeleteKey
- DeleteValue
- DynData
- EnumKey
- EnumValue
- Flush
- GetName
- GetSecurity
- GetSubkeyCount
- GetValueCount
- LoadKey
- LocalMachine
- NukeKey
- PerformanceData
- QueryDefValue
- QueryInfo
- QueryValue
- ReplaceKey
- Restore
- Save
- SetDefValue
- SetSecurity
- SetValue
- THandle
- UnLoadKey
- Users

Protected Member Functions
- Key
- Name
- ShouldClose
- SubkeyCount

ValueCount

Response Table Entries

TRegKey::TRegKey

TRegKey

Syntax 1

```
TRegKey(THandle baseKey, const char far* keyName, REGSAM samDesired =
  KEY_ALL_ACCESS, TCreateOK createOK = CreateOK);
```

Description

Creates or opens a key given a base key and a subkeyname. Security information is ignored in 16bit (and under Win95). This can also provide an ok-to-create or open-only indicator.

Syntax 2

```
TRegKey(const TRegKeyIterator& iter, REGSAM samDesired = KEY_ALL_ACCESS);
```

Description

Constructs a key given the current position of a regkey iterator.

Syntax 3

```
TRegKey(THandle aliasKey, bool shouldClose=false, const char far* keyName =
  0);
```

Description

Contructs a key that is an alias to an existing HKEY.

Syntax 4

```
~TRegKey();
```

Description

The destructor for this class.

TRegKey::ClassesRoot

TRegKey

Syntax

```
static TRegKey ClassesRoot;
```

Description

Special predefined root key used by shell and OLE applications.

TRegKey::ClassesRootClsid

TRegKey

Syntax

```
static TRegKey ClassesRootClsid;
```

Description

A subkey commonly used by shell and OLE applications.

TRegKey::CurrentConfig

TRegKey

Syntax

```
static TRegKey CurrentConfig;
```

Description

Special predefined root key.

TRegKey::CurrentUser

TRegKey

Syntax

```
static TRegKey CurrentUser;
```

Description

Special predefined root key defining the preferences of the current user.

TRegKey::DeleteKey

TRegKey

Syntax

```
long DeleteKey(const char far* subKeyName);
```

Description

Deletes the specified subkey of this registry key.

TRegKey::DeleteValue

TRegKey

Syntax

```
long DeleteValue(const char far* valName) const;
```

Description

Removes a named value from this registry key.

TRegKey::DynData

TRegKey

Syntax

```
static TRegKey DynData;
```

Description

Special predefined root key.

TRegKey::EnumKey

TRegKey

Syntax

```
long EnumKey(int index, char far* subKeyName, int subKeyNameSize) const;
```

Description

Enumerates the subkeys of this registry key.

TRegKey::EnumValue

TRegKey

Syntax

```
long EnumValue(int index, char far* valueName, uint32& valueNameSize,
  uint32* type=0, uint8* data=0, uint32* dataSize=0) const;
```

TRegKey::Flush

TRegKey

Syntax

```
long Flush() const;
```

Description

Writes the attribute of this key into the registry.

TRegKey::GetName

TRegKey

Syntax

```
const char far* GetName() const;
```

Description

Returns a string identifying this key.

TRegKey::GetSecurity

TRegKey

Syntax

```
long GetSecurity(SECURITY_INFORMATION secInf, PSECURITY_DESCRIPTOR secDesc,
  uint32* secDescSize);
```

Description

Retrieves a copy of the security descriptor protecting this registry key.

TRegKey::GetSubkeyCount

TRegKey

Syntax

```
uint32 GetSubkeyCount() const;
```

Description

Returns the number of subkeys attached to this key.

TRegKey::GetValueCount

TRegKey

Syntax

```
uint32 GetValueCount() const;
```

Description

Returns the number of values attached to this key.

TRegKey::LoadKey

TRegKey

Syntax

```
long LoadKey(const char far* subKeyName, const char far* fileName);
```

Description

Creates a subkey under HKEY_USER or HKEY_LOCAL_MACHINE and stores registration information from a specified file into that subkey. This registration information is in the form of a hive. A hive is a discrete body of keys, subkeys, and values that is rooted at the top of the registry hierarchy. A hive is backed by a single file and a .LOG file.

TRegKey::LocalMachine

TRegKey

Syntax

```
static TRegKey LocalMachine;
```

Description

Special predefined root key defining the physical state of the computer.

TRegKey::NukeKey

TRegKey

Syntax

```
long NukeKey(const char far* subKeyName);
```

Description

Completely eliminates a child key, including any of its subkeys. RegDeleteKey fails if a key has subkeys, so must tail-recurse to clean them up first.

TRegKey::PerformanceData

TRegKey

Syntax

```
static TRegKey PerformanceData;
```

Description

Special predefined root key used to obtain performance data.

TRegKey::QueryDefValue

TRegKey

Syntax

```
long QueryDefValue(const char far* subkeyName, char far* data, uint32*
  dataSize) const;
```

Description

Retrieves the default [unnamed] value associated with this key.

TRegKey::QueryInfo

TRegKey

Syntax

```
long QueryInfo(char far* class_, uint32* classSize, uint32* subkeyCount,
  uint32* maxSubkeySize, uint32* maxClassSize, uint32* valueCount, uint32*
  maxValueName, uint32* maxValueData, uint32* secDescSize, FILETIME far*
  lastWriteTime);
```

Description

Retrieves information about this registry key.

TRegKey::QueryValue

TRegKey

Syntax

```
long QueryValue(const char far* valName, uint32* type, uint8* data, uint32*
  dataSize) const;
```

Description

Retrieves the value associated with the unnamed value for this key in the registry.

TRegKey::ReplaceKey

TRegKey

Syntax

```
long ReplaceKey(const char far* subKeyName, const char far* newFileName,
  const char far* oldFileName);
```

Description

Replaces the file backing this key and all of its subkeys with another file, so that when the system is next started, the key and subkeys will have the values stored in the new file.

TRegKey::Restore

TRegKey

Syntax

```
long Restore(const char far* fileName, uint32 options=0);
```

Description

Reads the registry information in a specified file and copies it over this key. This registry information may be in the form of a key and multiple levels of subkeys.

TRegKey::Save

TRegKey

Syntax

```
long Save(const char far* fileName);
```

Description

Saves this key and all of its subkeys and values to the specified file.

TRegKey::SetDefValue

TRegKey

Syntax

```
long SetDefValue(const char far* subkeyName, uint32 type, const char far*
  data, uint32 dataSize);
```

Description

Sets the default [unnamed] value associated with this key.

TRegKey::SetSecurity

TRegKey

Syntax

```
long SetSecurity(SECURITY_INFORMATION secInf, PSECURITY_DESCRIPTOR secDesc);
```

Description

Sets the security descriptor of this key.

TRegKey::SetValue

TRegKey

Syntax 1

```
long SetValue(const char far* valName, uint32 data) const;
```

Description

Associates a 4-byte value with this key.

Syntax 2

```
long SetValue(const char far* valName, uint32 type, const uint8* data,
  uint32 dataSize) const;
```

Description

Associates a value with this key.

TRegKey::THandle

TRegKey

Syntax

```
operator THandle() const;
```

Description

Returns the HANDLE identifying this registry key.

TRegKey::UnLoadKey

TRegKey

Syntax

```
long UnLoadKey(const char far* subKeyName);
```

Description

Unloads this key and its subkeys from the registry.

TRegKey::Users

TRegKey

Syntax

```
static TRegKey Users;
```

Description

Special predefined root key defining the default user configuration.

TRegKey::Key

TRegKey

Syntax

```
THandle Key;
```

Description

This key's handle.

TRegKey::Name

TRegKey

Syntax

```
char far* Name;
```

Description

This key's name.

TRegKey::ShouldClose

TRegKey

Syntax

```
bool ShouldClose;
```

Description

Whether this key should be closed on destruction.

TRegKey::SubkeyCount

TRegKey

Syntax

```
uint32 SubkeyCount;
```

Description

The number of subkeys.

TRegKey::ValueCount

TRegKey

Syntax

```
uint32 ValueCount;
```

Description

The number of value entries.

TRegLink struct

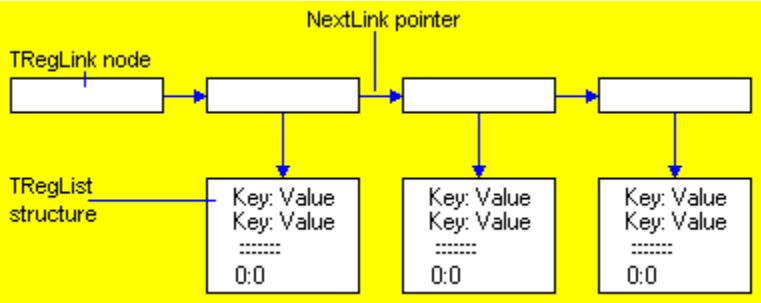
See also

Header File

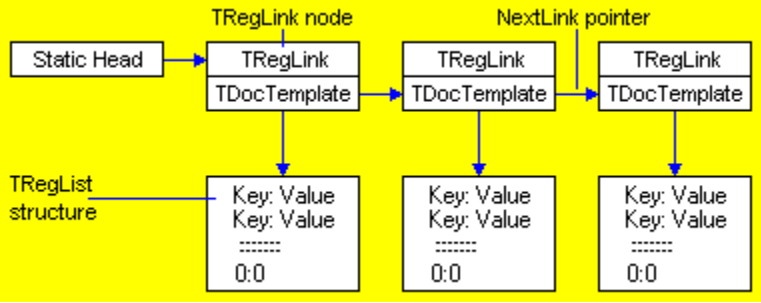
registry.h

Description

TRegLink is a linked structure in which each node points to a list of TRegList objects (or *TRegList*-derived objects) or TDocTemplate objects. Each object has an item name and a string value associated with the item name. The structure forms a typical linked list as the following diagram illustrates:



A TDocTemplate object uses the following variation of the *TRegLink* structure:



Public Constructors

TRegLink::TRegLink

Public Data Members

NextLink

Public Member Functions

AddLink
GetNext
GetRegList
RemoveLink

Protected Member Functions

Next
RegList

Protected Constructors

TRegLink

TRegLink::TRegLink

TRegLink

Syntax

```
TRegLink(TRegList& regList, TRegLink*& head);
```

Description

Constructs a reglink pointing to a reglist, and adds it to the end of the list.

Destructor

```
~TRegLink();
```

Description

The destructor for this class.

TRegLink::RegList

See also TRegLink

Syntax

```
TRegList* RegList;
```

Description

Points to the list of item value pairs.

TRegLink::AddLink

TRegLink

Syntax

```
static void AddLink(TRegLink*& head, TRegLink& newLink);
```

Description

Adds a new link to the end of the link list.

TRegLink::GetNext

TRegLink

Syntax

```
TRegLink* GetNext() const;
```

Description

Returns a pointer to the next link.

TRegLink::GetRegList

TRegLink

Syntax

```
TRegList& GetRegList() const;
```

Description

Returns a pointer to the registration parameter table (reglist).

TRegLink::RemoveLink

TRegLink

Syntax

```
static bool RemoveLink(TRegLink*& head, TRegLink& remLink);
```

Description

Removes a link from the link list. Returns true if the link is found and removed.

TRegLink::Next

TRegLink

Syntax

```
TRegLink* Next;
```

Description

The next RegLink.

TRegLink::NextLink

See also TRegLink

Syntax

```
TRegLink* NextLink;
```

Description

Points to the next link that references a list of items.

TRegList class

See also

Header File

locale .h

Description

Holds an array of items of type TRegItem. Provides functionality that lets you access each item in the array and return the name of the item. Instead of creating a *TRegList* directly, you can use ObjectWindows' registration macros to build a *TRegList* for you.

Public Data Members

Items

Public Constructors

TRegList::TRegList

Public Member Functions

Lookup

LookupRef

operator [ ]

TRegList::Items

See also TRegList

Syntax

```
TRegItem* Items;
```

Description

References the item value pair respectively; for example, *Prgid* and "My Sample Application".

TRegList::operator [ ]

TRegList

Syntax

```
const char* operator[](const char* key);
```

Description

The array operator uses an item name (*key*) to locate an item in the array.

TRegList::TRegList

See also TRegList

Syntax

```
TRegList(TRegItem* list):Items(List);
```

Description

Constructs a *TRegList* object from an array of TRegItems terminated by a NULL item name.

TRegList::Lookup

See also TRegList

Syntax

```
const char* Lookup(const char* key, TLangId lang =
  TLocaleString::UserDefaultLangId);
```

Description

Performs the lookup of the TRegItems using a *key* (an item name such as *progid*) and returns the value associated with the *key* (for example, "My Sample Application"). The value is returned in the language specified in *lang* (for example, French Canadian).

TRegList::LookupRef

See also TRegList

Syntax

```
TLocaleString& LookupRef(const char* key);
```

Description

Looks up and returns a reference to a local string value associated with a particular item name (*key*). You can then translate this string into the local language as necessary.

TRegValue class

Header File
registry.h

Description
TRegValue encapsulates a value-data entry within one registration key.

Public Constructors
- TRegValue::TRegValue

Public Member Functions
- Delete
- GetData
- GetDataSize
- GetDataType
- GetName
- operator *
- operator =
- RetrieveOnDemand
- Set
- uint32

TRegValue::TRegValue

TRegValue

Syntax 1

```
TRegValue(const TRegKey& key, const char far* name);
```

Description

Creates a registry value object from the specified registry key and name.

Syntax 2

```
TRegValue(const TRegValueIterator& iter);
```

Description

Creates a registry object from the current location of the specified iterator.

Destructor

```
~TRegValue();
```

Description

The destructor for this class.

TRegValue::operator *

TRegValue

Syntax

```
operator const char far*() const;
```

Description

Returns the data associated with this value as a const char*.

TRegValue::Delete

TRegValue

Syntax

```
long Delete();
```

Description

Removes this value from its associated key.

Note: The state of this value object becomes undefined.

TRegValue::GetData

TRegValue

Syntax

```
const uint8* GetData() const;
```

Description

Returns the type code for the data associated with this value.

TRegValue::GetDataSize

TRegValue

Syntax

```
const uint32 GetDataSize() const;
```

Description

Returns the size in bytes of the data associated with this value.

TRegValue::GetDataType

TRegValue

Syntax

```
const uint32 GetDataType() const;
```

Description

Returns the type code for the data associated with this value.

TRegValue::GetName

TRegValue

Syntax

```
const char far* GetName() const;
```

Description

Returns a string identifying this value.

TRegValue::operator =

TRegValue

Syntax 1

```
void operator =(const char far* v);
```

Description

Sets the data for this value to *v*.

Syntax 2

```
void operator =(uint32 v);
```

Description

Sets the data for this value to *v*.

TRegValue::RetrieveOnDemand

TRegValue

Syntax

```
void RetrieveOnDemand() const;
```

Description

Flag specifying whether data should be read upon construction of object or only when data is requested.

TRegValue::Set

TRegValue

Syntax 1

```
long Set(const char far* data);
```

Description

Sets the data associated with this value.

Syntax 2

```
long Set(uint32 data);
```

Description

Sets the data associated with this value.

Note: For 32-bit only.

Syntax 3

```
long Set(uint32 type, uint8* data, uint32 dataSize);
```

Description

Sets the data associated with this value. ‘type’ describes the type of the value. ‘data’ is the address of the data. ‘size’ specifies the length in characters.

TRegValue::uint32

TRegValue

Syntax

```
operator uint32() const;
```

Description

Returns the data associated with this value as a 32-bit unsigned integer.

TRegValueIterator class

Header File
registry.h

Description
TRegValueIterator is an iterator for walking through the values of a key.

Public Constructors
- TRegValueIterator::TRegValueIterator

Public Member Functions
- BaseKey
- Current
- operator ++
- operator --
- operator [ ]
- operator bool()
- Reset

TRegValueIterator::TRegValueIterator

TRegValueIterator

Syntax

```
TRegValueIterator(const TRegKey& regKey);
```

Description

Creates a subkey iterator for a registration key.

TRegValueIterator::BaseKey

TRegValueIterator

Syntax

```
const TRegKey& BaseKey() const;
```

Description

Returns the registration key that this iterator is bound to.

TRegValueIterator::operator bool()

TRegValueIterator

Syntax

```
operator bool();
```

Description

Tests the validity of this iterator. Returns **true** if the iterator's index is greater than or equal to 0 and less than the number of subkeys.

TRegValueIterator::Current

TRegValueIterator

Syntax

```
int Current() const;
```

Description

Returns the index to the current subkey.

TRegValueIterator::operator ++

TRegValueIterator

Syntax 1

```
uint32 operator ++();
```

Description

Pre-increments to the next value.

Syntax 2

```
uint32 operator ++(int);
```

Description

Post-increments to the next value

TRegValueIterator::operator --

TRegValueIterator

Syntax 1

```
uint32 operator --();
```

Description

Pre-decrements to the previous value.

Syntax 2

```
uint32 operator --(int);
```

Description

Post-decrements to the previous value.

TRegValueIterator::operator [ ]

TRegValueIterator

Syntax

```
uint32 operator [](int index);
```

Description

Sets the index of the iterator to the passed value. Return the new index.

TRegValueIterator::Reset

TRegValueIterator

Syntax

```
void Reset();
```

Description

Resets the value index to zero.

TResId class

Header File
geometry.h

Description
A simple support class, *TResId* creates a resource ID object from either an integer or an actual string identifier. For example, *TResId* encapsulates the use of *LPSTR (char_far*)* as a resource identifier. This resource identifier can be passed to various ObjectWindows classes. To handle these two different types of resource identifiers, *TResId* defines a conversion operator and provides two constructors that convert and use these native data types. One constructor accepts a 16-bit integer and the other accepts a character string.

Public Constructors
TResId::TResId

Public Member Functions
char far*
IsString

Friend Functions
operator >>
operator <<

TResId::TResId

TResId

Syntax 1

```
TResId();
```

Description

The default *TResId* constructor.

Syntax 2

```
TResId(int resNum);
```

Description

Creates a *TResId* object with the given *resNum*.

Syntax 3

```
TResId(const char far* resString);
```

Description

Creates a *TResId* object with the given *resString*.

TResId::IsString

TResId

Syntax

```
bool IsString() const;
```

Description

Returns **true** if this resource ID was created from a string; otherwise, returns **false**.

TResId::operator char far*()

TResId

Syntax

```
operator char far*();
```

Description

Typecasting operator that converts *Id* (a *TResId* private data member) to type **char far*** so that instances of *TResId* can be used in places where **char far*** data types are expected.

TResId::operator >>

See also TResId

Syntax

```
friend ipstream& operator >>(ipstream& is, TResId& id);
```

Description

Extracts a *TResId* object from *is* (the given input stream), and copies it to *id*. Returns a reference to the resulting stream, allowing the usual chaining of **>>** operations.

TResId::operator <<

See also TResId

Syntax 1

```
friend opstream& operator <<(opstream& os, const TResId& id);
```

Description

Inserts the given *TResId* object (*id*) into the *opstream* (*os*). Returns a reference to the resulting stream, allowing the usual chaining of **<<** operations.

Syntax 2

```
friend ostream& operator <<(ostream& os, const TResId& id);
```

Description

Formats and inserts the given *TResId* object (*id*) into the *ostream* (*os*). Returns a reference to the resulting stream, allowing the usual chaining of **<<** operations.

TResource class

Header File
wsyscls.h

Description
TResource simplifies access to a resource by encapsulating the find, load, lock and free steps for accessing a resource.

'T' represents a structure which defines the binary layout of the resource.

'resType' is a constant string that defines the resource type.

For example,

```
typedef TResource<DLGTEMPLATE, RT_DIALOG> TDlgResource;
TDlgResource dlgInfo(hInstance, IDD_ABOUTDLG);
DLGTEMPLATE* pDlgTmpl = dlgInfo;
```

Public Constructors
TResource::TResource

Public Member Functions
operator T*
IsOK

Protected Member Functions
MemHandle
MemPtr

TResource::TResource

TResource

Syntax 1

```
TResource(HINSTANCE hModule, TResId resId);
```

Description

Creates an object based on the module information and the resource identifier.

Syntax 2

```
TResource(HINSTANCE hModule, TResId resid, LANGID langid);
```

Description

Creates an object based on the module information and the resource identifier.

Destructor

```
~TResource();
```

Description

The destructor for this class.

TResource::T*

TResource

Syntax

```
operator T*();
```

Description

Conversion operator to point to the structure representing the binary layout of the resource.

TResource::IsOK

TResource

Syntax

```
bool IsOK() const;
```

Description

Confirms whether the resource was found.

TResource::MemHandle

TResource

Syntax

```
HGLOBAL MemHandle;
```

Description

The handle of the resource.

TResource::MemPtr

TResource

Syntax

```
T* MemPtr;
```

Description

Pointer to the locked resource.

TRgbQuad class

Header File
color.h

Description
TRgbQuad is a support class derived from the structure *tagRGBQUAD*, which is defined as follows:

```
typedef struct tagRGBQUAD {
   uint8  rgbBlue;
   uint8  rgbGreen;
   uint8  rgbRed;
   uint8  rgbReserved;
} RGBQUAD;
```

The elements *rgbBlue*, *rgbGreen*, and *rgbRed* specify the relative blue, green, and red intensities of a color. *rgbReserved* is not used and must be set to 0.

TRgbQuad is used in conjunction with the classes TPalette and *TColor* to simplify RGBQUAD-based color operations. Constructors are provided to create *TRgbQuad* objects from explicit RGB values, from *TColor* objects, or from other *TRgbQuad* objects.

Public Constructors
TRgbQuad::TRgbQuad

Public Member Functions
operator ==

TRgbQuad::TRgbQuad

TRgbQuad

Syntax 1

```
TRgbQuad(int r, int g, int b);
```

Description

Creates a *TRgbQuad* object with *rgbRed*, *rgbGreen*, and *rgbBlue* set to *r*, *g*, and *b* respectively. Sets *rgbReserved* to 0.

Syntax 2

```
TRgbQuad(TColor c);
```

Description

Creates a *TRgbQuad* object with *rgbRed*, *rgbGreen*, *rgbBlue* set to *c.Red*, *c.Green*, *c.Blue* respectively. Sets *rgbReserved* to 0.

Syntax 3

```
TRgbQuad(const RGBQUAD far& q);
```

Description

Creates a *TRgbQuad* object with the same values as the referenced RGBQUAD object.

TRgbQuad::operator ==

See also TRgbQuad

Syntax

```
bool operator ==(COLORREF cr) const;
```

Description

Returns true if the RGBQUAD has the same color components.

TRgbTriple class

Header File
color.h

Description
TRgbTriple is a support class derived from the structure *tagRgbTriple*, which is defined as follows:

```
typedef struct tagRGBTRIPLE {
    uint8  rgbBlue;
    uint8  rgbGreen;
    uint8  rgbRed;
} RGBTRIPLE;
```

The members *rgbBlue*, *rgbGreen*, and *rgbRed* specify the relative blue, green, and red intensities for a color.

TRgbTriple is used in conjunction with the classes TPalette and TColor to simplify bmci-color-based operations. Constructors are provided to create *TRgbTriple* objects from explicit RGB values, from *TColor* objects, or from other *TRgbTriple* objects.

Public Constructors
TRgbTriple::TRgbTriple

Public Member Functions
operator ==

TRgbTriple::TRgbTriple

See also

Syntax 1

```
TRgbTriple(int r, int g, int b);
```

Description

Creates a *TRgbTriple* object with *rgbRed*, *rgbGreen*, and *rgbBlue* set to *r*, *g*, and *b* respectively.

Syntax 2

```
TRgbTriple(TColor c);
```

Description

Creates a *TRgbTriple* object with *rgbRed*, *rgbGreen*, *rgbBlue* set to *c.Red*, *c.Green*, and *c.Blue* respectively.

Syntax 3

```
TRgbTriple(const RGBTRIPLE far& t);
```

Description

Creates a *TRgbTriple* object with the same values as the referenced RGBTRIPLE object.

TRgbTriple::operator ==

TRgbTriple

Syntax

```
bool operator ==(COLORREF cr) const;
```

Description

Returns true if the triple match color components.

TSize class

Header File
geometry.h

Description
TSize is a mathematical class derived from the structure *tagSIZE*.

The *tagSIZE* struct is defined as

```
struct tagSIZE {
   int cx;
   int cy;
};
```

TSize encapsulates the notion of a two-dimensional quantity that usually represents a displacement or the height and width of a rectangle. *TSize* inherits the two data members *cx* and *cy* from *tagSIZE*. Member functions and operators are provided for comparing, assigning, and manipulating sizes. Overloaded **<<** and **>>** operators allow chained insertion and extraction of *TSize* objects with streams.

Public Constructors
- TSize::TSize

Public Member Functions
- Magnitude
- operator +
- operator -
- operator ==
- operator !=
- operator +=
- operator -=
- operator >>
- operator <<
- X
- Y

TSize::TSize

See also TSize

Syntax 1

```
TSize();
```

Description

The default *TSize* constructor.

Syntax 2

```
TSize(int dx, int dy);
```

Description

Creates a *TSize* object with *cx* = *dx* and *cy* = *dy*.

Syntax 3

```
TSize(const POINT& point);
```

Description

Creates a *TSize* object with *cx* = *point.x* and cy = *point.y*.

Syntax 4

```
TSize(const SIZE& size);
```

Description

Creates a *TSize* object with *cx* = *size.cx* and *cy* = *size.cy*.

Syntax 5

```
TSize(uint32 dw);
```

Creates a *TSize* object with *cx* = LOWORD(*dw*) and *cy* = HIWORD(*dw*)).

TSize::Magnitude

TSize

Syntax

```
int Magnitude() const;
```

Description

Returns the length of the diagonal of the rectangle represented by this object. The value returned is an **int** approximation to the square root of ($cx^2 + cy^2$).

TSize::operator +

See also TSize

Syntax

```
TSize operator +(const TSize& size) const;
```

Description

Calculates an offset to this *TSize* object using the given *size* argument as the displacement. Returns the object (*cx* + *size.cx*, *cy* + *size.cy*). This *TSize* object is not changed.

TSize::operator -

See also TSize

Syntax 1

```
TSize operator -(const TSize& size) const;
```

Description

Calculates a negative offset to this *TSize* object using the given *size* argument as the displacement. Returns the point (*cx* - *size.cx*, *cy* - *size.cy*). This object is not changed.

Syntax 2

```
TSize operator -() const;
```

Description

Returns the *TSize* object (-*cx*, -*cy*). This object is not changed.

TSize::operator ==

See also TSize

Syntax

```
bool operator ==(const TSize& other) const;
```

Description

Returns **true** if this *TSize* object is equal to the *other TSize* object; otherwise returns **false**.

TSize::operator !=

See also TSize

Syntax

```
bool operator !=(const TSize& other) const;
```

Description

Returns **false** if this *TSize* object is equal to the *other TSize* object; otherwise returns **true**.

TSize::operator +=

See also TSize

Syntax

```
TSize& operator +=(const TSize& size);
```

Description

Offsets this *TSize* object by the given *size* argument. This *TSize* object is changed to (*cx* + *size.cx*, *cy* + *size.cy*). Returns a reference to this object.

TSize::operator -=

See also TSize

Syntax

```
TSize& operator -=(const TSize& size);
```

Description

Negatively offsets this *TSize* object by the given *size* argument. This object is changed to (*cx* - *size.cx*, *cy* - *size.cy*). Returns a reference to this object.

TSize::operator >>

See also TSize

Syntax

```
ipstream& operator >>(ipstream& is, TSize& s);
```

Description

Extracts a *TSize* object from *is*, the given input stream, and copies it to *s*. Returns a reference to the resulting stream, allowing the usual chaining of **>>** operations.

TSize::operator <<

See also TSize

Syntax 1

```
opstream& operator <<(opstream& os, const TSize& s);
```

Description

Inserts the given *TSize* object *s* into the *opstream os*. Returns a reference to the resulting stream, allowing the usual chaining of **<<** operations.

Syntax 2

```
ostream& operator <<(ostream& os, const TSize& s);
```

Description

Formats and inserts the given *TSize* object *s* into the *ostream os*. The format is "(*cx x cy*)". Returns a reference to the resulting stream, allowing the usual chaining of **<<** operations.

TSize::X

TSize

Syntax

```
int X() const;
```

Description

Returns the width.

TSize::Y

TSize

Syntax

```
int Y();
```

Description

Returns the height.

TString class

Header File
string.h

Description
TString is a flexible universal string envelope class. It facilitates efficient construction and assignment of many string types.

Public Constructors
- TString::TString

Public Member Functions
- GetLangId
- IsNull
- IsWide
- Length
- operator *
- operator =
- Relinquish
- RelinquishNarrow
- RelinquishSysStr
- RelinquishWide
- SetLangId

Protected Data Members
- S

TString::TString

TString

Syntax 1

```
TString(const char far* s = 0);
```

Description

Constructs a TString from a character array.

Syntax 2

```
TString(const wchar_t* s);
```

Description

Constructs a TString from a wide character array.

Syntax 3

```
TString(BSTR s, bool loan);
```

Description

Constructs a TString from a BSTR (OLE string).

Syntax 4

```
TString(TSysStr& s, bool loan);
```

Description

Constructs a TString from a system string (BSTR).

Syntax 5

```
TString(const string& s);
```

Description

Constructs a TString from a string.

Syntax 6

```
TString(TUString* s);
```

Description

Constructs a TString from a TUString.

Syntax 7

```
TString(const TString& src);
```

Description

Constructs a TString from a TString; this is the copy constructor.

Destructor

```
~TString();
```

Description

The destructor for this class.

TString::operator *

TString

Syntax 1

```
operator wchar_t*();
```

Description

Returns the string as a wchar_t*.

Syntax 2

```
operator const wchar_t*() const;
```

Description

Returns the string as a const wchar_t*.

Syntax 3

```
operator char*();
```

Description

Returns the string as a char*.

Syntax 4

```
operator const char far*() const;
```

Description

Returns the string as a const char far*.

TString::GetLangId

TString

Syntax

```
TLangId GetLangId();
```

Description

Gets the Language ID of this string.

TString::IsNull

TString

Syntax

```
bool IsNull() const;
```

Description

Returns true if the string is empty.

TString::IsWide

TString

Syntax

```
bool IsWide() const;
```

Description

Returns true if the string uses a wide character set.

TString::Length

TString

Syntax

```
int Length() const;
```

Description

Returns the length of the string.

TString::operator =

TString

Syntax 1

```
TString& operator =(wchar_t* s);
```

Description

Copies the contents of wchar_t* s into this string.

Syntax 2

```
TString& operator =(const wchar_t* s);
```

Description

Copies the contents of const wchar_t* s into this string.

Syntax 3

```
TString& operator =(char* s);
```

Description

Copies the contents of char* s into this string.

Syntax 4

```
TString& operator =(const char far* s);
```

Description

Copies the contents of const char* s into this string.

Syntax 5

```
TString& operator =(const string& s);
```

Description

Copies the contents of string s into this string.

Syntax 6

```
TString& operator =(const TString& s);
```

Description

Copies the contents of TString s into this string.

TString::Relinquish

TString

Syntax 1

```
wchar_t* Relinquish() const;
```

Description

Returns a pointer of type wchar_t to a copy of the string.

Syntax 2

```
char* Relinquish() const;
```

Description

Returns a pointer of type char to a copy of the string.

TString::RelinquishNarrow

TString

Syntax

```
char* RelinquishNarrow() const;
```

Description

Returns a pointer (char*) to a copy of the string.

TString::RelinquishSysStr

TString

Syntax

```
BSTR RelinquishSysStr() const;
```

Description

Returns a pointer (BSTR) to a copy of the string.

TString::RelinquishWide

TString

Syntax

```
wchar_t* RelinquishWide() const;
```

Description

Returns a pointer (wchar_t*) to a copy of the string.

TString::SetLangId

TString

Syntax

```
void SetLangId(TLangId id);
```

Description

Sets the language ID of this string.

TString::S

TString

Syntax

```
TUString* S;
```

Description

Instance of a helper object that implements functionality of a string object.

TXBase class

See also

Header File

except.h

Description

Derived from xmsg, *TXBase* is the base class for ObjectWindows and ObjectComponents exception-handling classes. The ObjectWindows classes that handle specific kinds of exceptions, such as out-of-memory or invalid window exceptions, are derived from TXOwl, which is in turn derived from *TXBase*. The ObjectComponents classes *TXOle* and *TXAuto* are derived directly from *TXBase*.

TXBase contains the functions *Clone* and *Throw*, which are overridden in all derived classes, as well as two constructors. The constructors increment InstanceCount, and the destructor decrements *InstanceCount*.

Public Constructors

TXBase::TXBase

Public Data Members

InstanceCount

Public Member Functions

Clone

Raise

Throw

TXBase::InstanceCount

TXBase

Syntax

```
static int InstanceCount;
```

Description

Counts the number of *TXBase* and *TXBase*-derived objects existing in a single application.

TXBase::TXBase

See also TXBase

Syntax 1

```
TXBase(const string& msg);
```

Description

Calls the xmsg class's constructor that takes a string parameter and initializes *xmsg* with the value of the string parameter.

Syntax 2

```
TXBase(const TXBase& src);
```

Description

Creates a copy of the *TXBase* object passed in the *TXBase* parameter.

Destructor

```
virtual ~TXBase;
```

Description

Destroys the *TXBase* object and decrements the InstanceCount data member.

TXBase::Clone

TXBase

Syntax

```
virtual TXBase* Clone();
```

Description

Makes a copy of the exception object.

TXBase::Raise

TXBase

Syntax

```
static void Raise(const string& msg);
```

Description

Constructs a TXBase exception from scratch, and throws it.

TXBase::Throw

TXBase

Syntax

```
virtual void Throw();
```

Description

Throws the exception object.

TXRegistry class

Header File
registry.h

Description
TXRegistry is the object thrown when exceptions are encountered within the WinSys Registry classes.

Public Constructors
- TXRegistry::TXRegistry

Public Member Functions
- Check
- Key

TXRegistry::TXRegistry

TXRegistry

Syntax 1

```
TXRegistry(const TXRegistry& copy);
```

Description

The copy constructor. Constructs a new registry exception object by copying the one passed as `copy`.

Syntax 2

```
TXRegistry(const char* msg, const char* key);
```

Description

Creates a registry exception object. `msg` points to an error message, and `key` points to the name of the registry key that ObjectComponents was processing when the exception occurred.

TXRegistry::Check

TXRegistry

Syntax

```
static void Check(long stat, const char* key);
```

Description

Registry exception checking. Throws a TXRegistry if the argument is non-zero.

TXRegistry::Key

TXRegistry

Syntax

```
const char* Key;
```

Description

Identifies the registry which caused the exception.