## D:\dev\Projet\CIsiLcdDisplayDemo\Doc\annim.gif

## Introduction

I’m an Industrial Development Software Supervisor that is the reason why I need controls able to display values clearly. 7 or16 segments control is good for that. I based this controls on GDI+ library for 2 reasons, I wanted a nice GUI and I am interested in learning about this library.

Many articles from CodeProject helped me to write this control (I can’t remember all of them). Many thanks to all the Code Project community and special thanks to [**Ambalavanar Thirugnanam**](http://www.codeproject.com/script/Membership/View.aspx?mid=548300)for his excellent article: <http://www.codeproject.com/KB/GDI-plus/AquaGauge.aspx>.

I Hope this article will be able to help you in your future project. All modifications, derived work, or anything else able to improve this control are welcome, please let me known about them.

**Features**

* Full **resizable** control.
* Display numeric (signed or not) or text value.
* Glossy and light effect (to have a nice look ☺ ).
* Double buffering (no flicking).
* Draw transparent.
* Blinking.
* Segments (or dots) size adjustment.

## Using the CIsiLcdDisplay Control

**Step 1.** This control uses the GDI+ library, before you use it, you must initialize this library. If your application uses another GDI+ components you probably already did that otherwise you could use the following macro to initialize, friendly, the GDI+ library.

In your CWinApp derived class header

class MyApp : public CWinApp

{

Private :

DECLARE\_GDI\_PLUS;

…//

**Step 2.** Implement the control

1. Insert all .h and .cpp files into your project.
2. Insert the Font5x7.bmp file as a bitmap resource.
3. Add a static **control** into your dialog resource.
4. Add a member variable for that static **control**.
5. Modify the variable declaration.  
   Change this:

CStatic m\_MyLcdCtrl

To this:

CIsiLcdDisplay m\_MyLcdCtrl

1. Don't forget the #include at the top of the include file:

#include "IsiLcdDisplay.h"

**Step 2Bis.** Dynamic create.

If you want to dynamically create this control, please insert the following code in the OnInitDialog method:

// CIsiLcdDisplay m\_MyLcdCtrl

CRect lCRect(x,y,dx,dy); //Size and position of the control in the dialog

if ( !m\_MyLcdCtrl.Create(\_T(“”),WS\_CHILD|WS\_VISIBLE, lCRect, this, -1 ) )

{

ASSERT( 0 ); // Something wrong

}

**Step 3.** Set the look of your control

In the OnInitDialog() method add the following lines according with the desired design.

m\_CIsiLcdSample14.LoadMatrix( IDB\_MATRIX5x7 ); // Load 5x7 matrix

m\_CIsiLcdSample14.SetNbSegment( MatrixDot, FALSE );// Select matrix mode

// (dot)

m\_CIsiLcdSample14.SetValueNbDigit( 6,2 ); // 6 Digits, precision 2

m\_CIsiLcdSample14.ShowSign( TRUE ); // Show sign

m\_CIsiLcdSample14.SetColor( RGB(0,150,0) ); // Set green color (~50%)

**Step 4.** Update the value to be displayed.

If the StringMode is enabled, you have to use the SetWindowText( \_T(“New test”) ); method to update the control otherwise use the SetValue( dNewVal ) method.

m\_CIsiLcdSample14.SetValue( 2.35 ); // Set value to be displayed

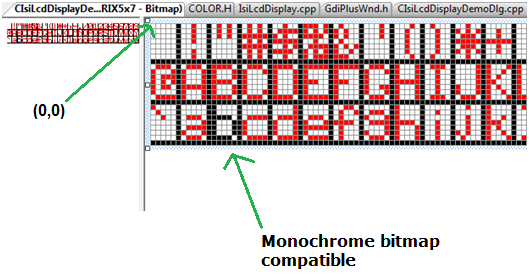
Or

//CIsiLcdSample14.SetTextmode( TRUE );

m\_CIsiLcdSample14.SetWindowText( \_T(“My text”); // Set value to be displayed

Monochrome bitmap compatible

**X \* Y Matrix file format.**



The matrix is stored in a bitmap file. I chose this format because a 5\*7 matrix with 96 characters is too big to be edited manually (look at the 16segments arrays I let you imagine the dimension of a 20x20 matrix). The bitmap is a good file format because you can find a lot of editors and you can edit it directly in visual studio. It’s not necessary to develop a specific editor to create new font matrix. You can use 2,16,256 or true color bitmap, but 16 colors bitmap seems to be the best format because we need only 3 different colors (2 colors is enough but edit the matrix with 3 colors is easier). 1 color is used to identify OFF pixels, 1 for ON pixels and the last one to make the difference between each character. The bitmap is organized at 3 lines with 32 characters, you must respect this structure. The maximum character size is fixed to 20x20 because I need a limit to avoid crash during bitmap analyze.

How is analyzed the bitmap? The first printable character is the space char. All pixels of this character should be OFF, so the first pixel (at the position 0,0 ) must be the background color. After that I have to known the size of the matrix char (you are not limited in a 5\*7 matrix). I parse the first line until background color is found. I use the same technique to calculate the high of font matrix.

**Create your matrix.**

**Step1:** Create a new bitmap (with = x \* 32 +32, high = y \* 3 + 3) or edit mine.

x,y are the dimension of each character. With a 5\*7 matrix the bitmap size is 192\*24.

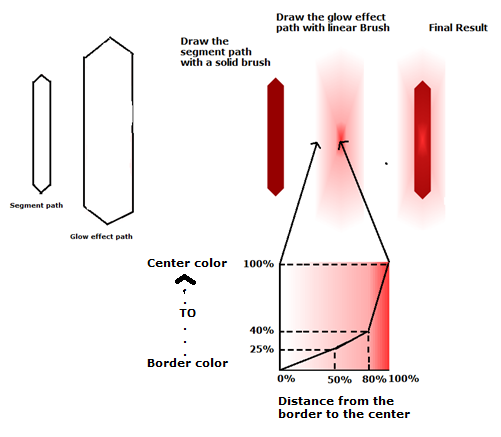
If the bitmap doesn’t respect this formula, the LoadMatrix() method will fail.

**Step2:** Draw lines separations

**Step3:** Draw characters

**Step4:** Load you new matrix with the LoadMatrix method.

**Points of Interest**

The light effect gives to the control a nice look (in my opinion, if you don’t like the effect you can disable it). To do it, it’s really easy.

Step 1 Create Fill the segment path with the ON segment color.

Step 2 Scale up a little bit the segment path to do the glow effect path

// PointF SegmentH[ 6 ]; // Horizontal segment

//////////////////////////////////////////////////

Matrix MatrixGlowH;

MatrixGlowVH.Scale( GLOW\_SCALE\_Y,GLOW\_SCALE\_X );

// Create vertical segment path ///////////////////////////////////////////////

m\_pathSegmentH.AddPolygon(SegmentH, 6);

m\_pathSegmentH\_Glow.AddPolygon(SegmentH, 6);

m\_pathSegmentH\_Glow.Transform( &MatrixGlowV );

Step 3 Create the linear bush based on the glow effect path.

// Set blend factors and positions for the path gradient brush.

REAL fac[] = {

0.0f, // 0 percent

0.25f, // 25 percent

0.4f, // 40 percent

1.0f};

REAL posfac[] = {

0.0f, // 50 percent

0.5f, // 50 percent

0.8f, // 80 percent

1.0f};

// Create glow brush //////////////////////////////////////////////

PathGlow->Transform( &TmpMatrix );

PathGradientBrush PathBrushGlow( PathGlow );

PathBrushGlow.SetCenterColor( Clr(GetGlowColor()).value);

Color OutColor(0,0,0,0);

int iNbColor = 1;

PathBrushGlow.SetSurroundColors( &OutColor, &iNbColor );

PathBrushGlow.SetBlend(fac, posfac, 4);

Step 4 Fill the path with the brush

pGraphics->FillPath( &PathBrushGlow, PathGlow);

## CIsiLcdDisplay inherits from CIsiCompBase ?

Yes the CIsiLcdDisplay inherits from CIsiCompBase class because I have other industrial controls and I don’t want to have to rewrite the same code. The CIsiCompBase has been designed to develop industrial gauges or displayers. It provides you a double buffering system to paint your control and other methods dedicated to this kind of control. I think it could be an interesting base for your future projects. If you need only the double buffering mechanism, you should use the template *CGdiPlusWnd* in file GdiPlusWnd.h. Why a template? Because the template allows you to create controls based on any CWnd objects (like CButton,CStatic, CSliderCtrl…).

Using the CIsiCompBase of CGdiPlusWnd template to create your gauges, what you have to do ?

**Step 1. Draw the back ground:**

**You have to overload the** RebuildGDIbkGnd() method.

virtual BOOL RebuildGDIbkGnd(Graphics\* pGraphics, CRect &WindowRect)

{

// Draw the back ground or your control in the pGraphics object.

// You can draw a complex image (and take lot of time) because this method is call

// only when the control is moved, resized, or displayed for the first time.

// Usually this method is called only one time.

// CRect &WindowRect provides to you the size of you control

return TRUE;

}

**Step 2. Draw the pointer, bare or anything else to display the value of you control.**

**You have to overload the** OnPaint() method.

virtual BOOL OnPaint(Graphics\* pGraphics, CRect &WindowRect)

{

// Draw the mobiles part of you control

// At the end you should add the glossiness if you control has this option

return TRUE;

}

**Documentation**

## Class methods

void SetBlink( int iPeriod );

|  |
| --- |
| Set the blinking period in ms. If this parameter is 0 (default value) the control will not blink. |

int GetBlink();

|  |
| --- |
| Return current blink period. |

void ShowOffSegment(BOOL bShow, BOOL bRedraw = TRUE);

|  |
| --- |
| ***bShow***: Specifies whether to show or hide off segments. If this parameter is TRUE, OFF segments are shown. If the parameter is FALSE, OFF segments are hidden. Default value is TRUE.  ***bRedraw:*** If TRUE redraw immediately the control. |

int IsShowOffSegmentVisible();

|  |
| --- |
| Return TRUE if OFF segments are shown, otherwise return FALSE.  ***bRedraw:*** If TRUE redraw immediately the control. |

void StringMode(BOOL bString, BOOL bRedraw = TRUE);

|  |
| --- |
| ***bString:*** TRUE if the control must be updated with the SetWindowText() .method. FALSE if the control must be update with SetValue() method. Default value FALSE.  ***bRedraw:*** If TRUE redraw immediately the control. |

BOOL IsStringMode();

|  |
| --- |
| *Return current mode*. |

void ShowSign(BOOL bShow, BOOL bRedraw = TRUE);

|  |
| --- |
| ***bShow***: Specifies whether to show or hide the sign. If this parameter is TRUE, OFF sign is shown. If the parameter is FALSE, OFF sign are hidden. Default value is FALSE. This method as no effect if StringMode() is enabled.  ***bRedraw:*** If TRUE redraw immediately the control. |

BOOL ISShowSign();

|  |
| --- |
| Return TRUE if sign is shown, otherwise return FALSE. |

void HexMode(BOOL bHex, BOOL bRedraw = TRUE);

|  |
| --- |
| ***bHex***: Specifies if value must be displayed in hexadecimal format. If this parameter is TRUE, the value is displayed in hexadecimal mode, otherwise decimal is used. Default value is FALSE. This method as no effect if StringMode() is enabled.  ***bRedraw:*** If TRUE redraw immediately the control. |

BOOL IsHexMode();

|  |
| --- |
| Return TRUE if value is displayed in hexadecimal format, otherwise return FALSE. |

void EnableValue( BOOL bEnable );

|  |
| --- |
| ***bEnable***: Specifies if the current value is valid or not. If this parameter is FALSE all segments are OFF. Default value is TRUE. |

BOOL IsEnableValue();

|  |
| --- |
| Return TRUE if current value is enabled. |

BOOL SetSegmentSize( float fSize, BOOL bRedraw = TRUE );

|  |
| --- |
| ***fSize***: .Segments scale factor. 0.5f means 50%, 1.5f means 150%... Default value 1.0f.  ***bRedraw:*** If TRUE redraw immediately the control.  This method allows to you to adjust the segments size. |

float GetSegmentSize( );

|  |
| --- |
| Return the current segments scale factor. |

void SetColor (COLORREF cOn,COLORREF cOff = 0, COLORREF cGlow = 0);

|  |
| --- |
| ***cOn***. Color when the segments are ON.  ***cOff.*** Color when the segments are OFF. If this parameter is 0 the OFF color is atomically calculated from the ON color.  ***cGlow:*** Color of the light effect. If this parameter is 0, the color is atomically calculated from the ON color. If this parameter is RGB(255,255,255) (white) the light effect is disabled. |

COLOREEF GetColorSegmentOn( );

|  |
| --- |
| Return color when segments are ON. |

COLOREEF GetColorSegmentOff( );

|  |
| --- |
| Return color when segments are OFF. |

COLOREEF GetColorSegmentGlow( );

|  |
| --- |
| Return color of the light effect. |

vois SetSegmentStyle( SegmentStyle Style, BOOL bRedraw = TRUE );

|  |
| --- |
| ***Style***: Set Style of segments.  *Segment7*: 7 Segments style  *Segment16*:16 Segemnts style  *MatrixDot* X\*Y Dot matrix (Default 5x7)  *MatrixSquare* X\*Y Square matrix (Default 5x7)  ***bRedraw:*** If TRUE redraw immediately the control.  **Important:** When using style ***MatrixDot*** or ***MatrixSquare***, you must load a font matrix before. Please see the LoadMatrix() method below. |
|  |
|  |

SegmentsStyle GetSegmentStyle();

|  |
| --- |
| Return current segments style. |

void SetScrollSpeed( int iSpeed );

|  |
| --- |
| ***iSpeed***. Delay in ms between each move. If this parameter is 0 the scrolling is disabled. |

int GetScrollSpeed();

|  |
| --- |
| Return current scrolling speed. |

BOOL LoadMatrix ( UINT uiRes);

|  |
| --- |
| ***uiRes:*** Contains the ID of the bitmap matrix resource.  Return nonzero if success, otherwise 0. |

BOOL LoadMatrix ( LPCTSTR lpzFileName);

|  |
| --- |
| ***lpzfileName:*** A string that is the path of the matrix bitmap file.  Return nonzero if success, otherwise 0. |

BOOL LoadMatrix( CBitmap \*pBitmap);

|  |
| --- |
| ***\*pBitmap:*** A CBitmap object that is the matrix to be loaded.  Return nonzero if success, otherwise 0. |

## Based class methods

void SetValueNbDigit (int iNbDigit, int iPrecision, BOOL bRedraw= TRUE);

|  |
| --- |
| ***iNbDigit***: .Number total of digit to be displayed.  ***iPrecision***: .Number of digit after the ( , ).  ***bRedraw:*** If TRUE redraw immediately the control. |

void SetValueNbDigit (int iNbDigit, int iPrecision, BOOL bRedraw= TRUE);

|  |
| --- |
| ***iNbDigit***: .Number total of digit to be displayed.  ***iPrecision***: .Number of digit after the ( , ).  ***bRedraw:*** If TRUE redraw immediately the control.  **Important:** This method as no effect is the StringMode is enabled. |

int GetValueNbDigit ();

|  |
| --- |
| Return the number of digit displayed. |

int GetValuePrecision ();

|  |
| --- |
| Return the current precision of the control. |

void Transparent( BOOL bTransparent, BOOL bRedraw= TRUE);

|  |
| --- |
| ***Tranparent***: .If this parameter is TRUE, the background of the control is not drawn. Default value is FALSE.  ***bRedraw:*** If TRUE redraw immediately the control. |

BOOL GetTransparent( );

|  |
| --- |
| Return current transparent drawing mode. |

void ShowGlass( BYTE bShow, BOOL bRedraw= TRUE);

|  |
| --- |
| ***bShow***: .Level [0;255] of the glassy effect. If this parameter is 0, the glassy effect is disabled.  ***bRedraw:*** If TRUE redraw immediately the control. |

BYTE GetShowGlass );

|  |
| --- |
| Return current level of the glassy effect. |

void SetRoundBorder(int iRound, BOOL bRedraw = TRUE);

|  |
| --- |
| ***iRound***: .Round corners of the control.  ***bRedraw:*** If TRUE redraw immediately the control. |

int GetRoundBorder( );

|  |
| --- |
| Return current round border value. |

void SetBkgndColor (COLORREF c, BOOL bRedraw = TRUE);

|  |
| --- |
| ***c***: .Background color. This parameter as no effect when drawing transparent.  ***bRedraw:*** If TRUE redraw immediately the control. |

COLORREF GetBkgndColor ( );

|  |
| --- |
| Return Background color. |

**Compatibility**

**This control has been developed under VS2005 but should be compatible with VS6 and of course VS20008 and 2010.**

**OS Compatibility**

**This control is compatible with 2000, XP, Vista, Seven (tested) and should be compatible with Window Mobile 6 (not tested).**

**Important:** Under Windows2000, you must copy the GdiPlus.dll file in the program directory. You can easily find this DLL on the web.

**History**

* Version 1.0 - Initial Version