.NET GDI+

Overview
- GDI+ namespaces
- Size, Point, Rectangle, and Region
- System.Drawing.Graphics
- Examples

References
# Core GDI+ Namespaces

<table>
<thead>
<tr>
<th>GDI+ Namespace</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.Drawing</td>
<td>This is the core GDI+ namespace, which defines numerous types for basic rendering as well as the Graphics type.</td>
</tr>
<tr>
<td>System.Drawing.Drawing2D</td>
<td>This namespace offers types used for more advanced 2D graphics functionality.</td>
</tr>
<tr>
<td>System.Drawing.Imaging</td>
<td>This namespace defines types that allow one to directly manipulate images.</td>
</tr>
<tr>
<td>System.Drawing.Printing</td>
<td>This namespace defines types that allow one to render images for printing.</td>
</tr>
<tr>
<td>System.Drawing.Text</td>
<td>This namespace allows one to manipulate collections of fonts.</td>
</tr>
</tbody>
</table>
Point(F)

<table>
<thead>
<tr>
<th>Member</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>These operators allow one to manipulate the underlying (x, y) point.</td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
<tr>
<td>==</td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>These properties allow one to get and set the underlying (x, y) values.</td>
</tr>
<tr>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>IsEmpty</td>
<td>This property returns true if X and Y are both set to zero.</td>
</tr>
<tr>
<td>Offset()</td>
<td>This method translates a given Point type by a given amount.</td>
</tr>
</tbody>
</table>
Rectangle(F)

<table>
<thead>
<tr>
<th>Member</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>These operators allow one to test whether two rectangles have identical values or not.</td>
</tr>
<tr>
<td>!=</td>
<td></td>
</tr>
<tr>
<td>Inflate(), Intersect(), Union()</td>
<td>These static methods allow one to expand a rectangle, or create a new rectangle as a result of an intersection of union operation.</td>
</tr>
<tr>
<td>Top, Left, Bottom, Right</td>
<td></td>
</tr>
<tr>
<td>Height, Width</td>
<td>Height and width of a rectangle.</td>
</tr>
<tr>
<td>Contains()</td>
<td>This method can be used to determine whether a given Point is within the bounds of a rectangle.</td>
</tr>
<tr>
<td>X, Y</td>
<td>These properties return the x and y coordinates of the Rectangle’s upper left corner.</td>
</tr>
</tbody>
</table>
These properties allow one to get and set the current dimension of a Size type.

<table>
<thead>
<tr>
<th>Member</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>These operators allow one to manipulate the underlying Size type.</td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
<tr>
<td>==</td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td></td>
</tr>
<tr>
<td>Height</td>
<td></td>
</tr>
<tr>
<td>Width</td>
<td></td>
</tr>
</tbody>
</table>

These properties allow one to get and set the current dimension of a Size type.
# Region

<table>
<thead>
<tr>
<th>Member</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complement()</td>
<td>Updates this Region to the portion of the specified Graphics object that does not intersect with this Region.</td>
</tr>
<tr>
<td>Exclude()</td>
<td>Updates this Region to the portion of its interior that does not intersect with the specified Graphics object.</td>
</tr>
<tr>
<td>GetBounds()</td>
<td>Returns a RectangleF that represents a rectangular region that bounds this Region.</td>
</tr>
<tr>
<td>Intersect()</td>
<td>Updates this Region to the intersection of itself with the specified Graphics object.</td>
</tr>
<tr>
<td>Translate()</td>
<td>Offsets the coordinates of this Region by the specified amount.</td>
</tr>
<tr>
<td>Union()</td>
<td>Updates this Region of the union minus the intersection of itself with the specified Graphics object.</td>
</tr>
</tbody>
</table>
public class MainForm : Form
{
    protected override void OnPaint(PaintEventArgs e)
    {
        Graphics lGraphics = e.Graphics; // get HDC

        lGraphics.DrawString( "Hello World!", new Font( "Tacoma", 28 ), new SolidBrush( Color.Blue ), 16, 100 ); // text, font, brush, Point.X, Point.Y
    }
}
Output

(16,100)

Hello World!
public class MainForm : Form
{
    public MainForm()
    {
        Paint += new PaintEventHandler( DoPaint );
    }

    private void DoPaint( object sender, PaintEventArgs e )
    {
        Graphics lGraphics = e.Graphics;                             // get HDC
        lGraphics.DrawString( "Hello World!",                      // text
                                new Font( "Tacoma", 28 ),     // font
                                new SolidBrush( Color.Blue ), // brush
                                16,                     // Point.X
                                100 );                  // Point.Y
    }
}
Output

(16,100)

Hello World!
## Graphics

<table>
<thead>
<tr>
<th>Member</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FromHDC(), FromHandle(), FromImage()</td>
<td>These static methods provide a way to obtain a valid Graphics object from a given image or GUI widget. These methods return a canvas for drawing.</td>
</tr>
<tr>
<td>Clear()</td>
<td>Fills a Graphics object with a specified color, erasing the current drawing surface.</td>
</tr>
<tr>
<td>DrawArc(), DrawEllipse(), DrawLine(), DrawLine(), DrawLine(), DrawRectangle()</td>
<td>These methods (among others) are used to render a given image or geometric pattern.</td>
</tr>
<tr>
<td>FillEllipse(), FillPie(), FillRectangle()</td>
<td>These methods (among others) are used to fill the interior of a given geometric shape.</td>
</tr>
</tbody>
</table>
Coordinate System

(0, 0)

X (width)

Y (height)

upper left corner
public class MainForm : Form
{
    private void DoPaint( object sender, PaintEventArgs e )
    {
        Graphics lGraphics = e.Graphics;  // get HDC

        lGraphics.DrawRectangle( new Pen( Color.Red ),  // pen
                                 10,                // X
                                 10,                // Y
                                 100,               // width
                                 150 );             // height
    }
}
Output

(10,10)
public class MainForm : Form
{
    private void DoPaint( object sender, PaintEventArgs e )
    {
        Graphics lGraphics = e.Graphics; // get HDC
        Image lImage = new Bitmap( @"c:\WINNT\Blue Lace 16.bmp" );
        Brush lBrush = new TextureBrush( lImage );

        lGraphics.DrawRectangle( new Pen( lBrush, 20.0F ), // pen
                                10,                // X
                                10,                // Y
                                100,               // width
                                150 );             // height
    }
}
Output

Com S 430
## GraphicsUnit

<table>
<thead>
<tr>
<th>Enumeration Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display</td>
<td>Specifies 1/75 inch as the unit of measure.</td>
</tr>
<tr>
<td>Document</td>
<td>Specifies the document unit (1/300 inch) as the unit of measure.</td>
</tr>
<tr>
<td>Inch</td>
<td>Specifies the inch as the unit of measure.</td>
</tr>
<tr>
<td>Millimeter</td>
<td>Specifies the millimeter as the unit of measure.</td>
</tr>
<tr>
<td>Pixel</td>
<td>Specifies a device pixel as the unit of measure.</td>
</tr>
<tr>
<td>Point</td>
<td>Specifies a printer’s point (1/72 inch) as the unit of measure.</td>
</tr>
</tbody>
</table>
Color Dialog
GDI + Test illustrates some of the capabilities of Graphics.
// MenuItem (Line) Event handler
private void menuItem18_Click(object sender, System.EventArgs e)
{
    // acquire HDC using the Form's Handle property
    Graphics lGraphics = Graphics.FromHwnd( this.Handle );
    // set selected measurement
    lGraphics.PageUnit = fMeasurement;
    // create a pen using selected color
    Pen lPen = new Pen( fDefaultColor );
    // draw a line from fOrigin to (200, 200)
    lGraphics.DrawLine( lPen, fOrigin, new Point( 200, 200 ) );
}
Example

- Inch
- Origin (0, 0)
- Color.Black
// MenuItem (Rectangle) Event handler
private void menuItem16_Click(object sender, System.EventArgs e)
{
    // acquire HDC using the Form’s Handle property
    Graphics lGraphics = Graphics.FromHwnd(this.Handle);
    // set selected measurement
    lGraphics.PageUnit = fMeasurement;
    // create a pen using selected color
    Pen lPen = new Pen(fDefaultColor);
    // draw a rectangle from fOrigin to (150, 150)
}
Example

- Millimeter
- Origin (16, 16)
- Color.Blue
// MenuItem (Arc) Event handler
private void menuItem19_Click(object sender, System.EventArgs e)
{
    // acquire HDC using the Form’s Handle property
    Graphics lGraphics = Graphics.FromHwnd( this.Handle );
    // set selected measurement
    lGraphics.PageUnit = fMeasurement;
    // create a pen using selected color
    Pen IPen = new Pen( fDefaultColor );
    // draw an arc from fOrigin with size (200, 200)
    lGraphics.DrawArc( IPen,
                        new Rectangle( fOrigin, new Size( 200, 200 ) ),
                        90.0F, 280.0F ); // startAngle, sweepAngle
}
Example

- Pixel
- Origin (100, 100)
- Color.Red
// MenuItem (Pie) Event handler
private void menuItem20_Click(object sender, System.EventArgs e)
{
    // acquire HDC using the Form's Handle property
    Graphics lGraphics = Graphics.FromHwnd( this.Handle );
    // set selected measurement
    lGraphics.PageUnit = fMeasurement;
    // create a pen using selected color
    Pen lPen = new Pen( fDefaultColor );
    // draw a pie from fOrigin with size (200, 200)
                        30, 300 ); // startAngle, sweepAngle
}
Example

- Pixel
- Origin (100, 100)
- Color: Red
// MenuItem (Fill Pie) Event handler
private void menuItem21_Click(object sender, System.EventArgs e)
{
    // acquire HDC using the Form's Handle property
    Graphics lGraphics = Graphics.FromHwnd( this.Handle );
    // set selected measurement
    lGraphics.PageUnit = fMeasurement;
    // create a brush using selected color
    Brush lBrush = new SolidBrush( fDefaultColor );
    // draw a filled pie from fOrigin with size (150, 150)
        60.0F, 245.0F ); // startAngle, sweepAngle
}

Example

- Millimeter
- Origin (16, 16)
- Color: Blue
DrawString

// MenuItem (Text) Event handler
private void menuItem22_Click(object sender, System.EventArgs e)
{
    // acquire HDC using the Form's Handle property
    Graphics lGraphics = Graphics.FromHwnd(this.Handle);
    // set selected measurement
    lGraphics.PageUnit = fMeasurement;
    // draw a string from fOrigin using Tacoma, 14pt
    lGraphics.DrawString("This is a GDI + Text string!",
                          new Font("Tacoma", 14 ),
                          new SolidBrush(fDefaultColor ),
                          fOrigin );
}

Example

This is a GDI+ Text string!

- Millimeter
- Origin (16, 16)
- Color.Purple
Drawstring With Texture

// MenuItem (Textured Text) Event handler
private void menuItem23_Click(object sender, System.EventArgs e)
{
    // acquire HDC using the Form’s Handle property
    Graphics lGraphics = Graphics.FromHwnd( this.Handle );
    // set selected measurement
    lGraphics.PageUnit = fMeasurement;
    // draw a textured string from fOrigin using Tacoma, 44pt
    Image lTextureBrushImage =
        new Bitmap( @"c:\WINNT\" + fTextureName );
    Brush lBrush = new TextureBrush( lTextureBrushImage );
    lGraphics.DrawString( "This is a GDI+ Text string!",
        new Font( "Tacoma", 44 ),
        lBrush, fOrigin );
}
Example

This is a GDI+

- Inch
- Origin (0, 0)
- Feather
Building a User Control

- We want to define a simple user control that provides a graphical user interface to a counter with a critical region.

- The counter is represented by a pie shape that grows over time.

- The critical section is also represented by a pie shape. The user should be able to change the bounds of the critical section using the mouse.

- If the counter enters the critical section, then the control must raise an “EnterSection” event. Similarly, if the counter leaves the critical section, then the control must raise an “LeaveSection” event.
Design

Panel

Label

Critical Section

270

Leave Critical Section

180

Counter

90

Enter Critical Section

Com S 430
Runtime
Graphical Design

Dock = Top
Dock = Left
Dock = Fill
Dock = Bottom
Dock = Right

Com S 430
{
    private System.Windows.Forms.Label fNorthLabel;
    private System.Windows.Forms.Label fEastLabel;
    private System.Windows.Forms.Label fSouthLabel;
    private System.Windows.Forms.Label fWestLabel;
}
Component Configuration

```c
class MyClass
{
    private:
        Label fNorthLabel;
        Label fEastLabel;
        Label fSouthLabel;
        Label fWestLabel;
        Panel fDrawingPanel;

does something

    private:
        void InitializeControl()
        {
            // create elements
            fNorthLabel = new Label();
            fEastLabel = new Label();
            fSouthLabel = new Label();
            fWestLabel = new Label();
            fDrawingPanel = new Panel();
            // suspend layout
            fNorthLabel.SuspendLayout();
            fEastLabel.SuspendLayout();
            fSouthLabel.SuspendLayout();
            fWestLabel.SuspendLayout();
            fDrawingPanel.SuspendLayout();
            // resume layout
            fNorthLabel.ResumeLayout(false);
            fEastLabel.ResumeLayout(false);
            fSouthLabel.ResumeLayout(false);
            fWestLabel.ResumeLayout(false);
            fDrawingPanel.ResumeLayout(false);
        }
};
```
Label Configuration

// fEastLabel
fEastLabel.Dock = DockStyle.Right;
fEastLabel.ForeColor = Color.Blue;

// use ClientRectangle as reference size
fEastLabel.Location = new Point( ClientRectangle.Width - 25, 25 );
fEastLabel.Size = new Size( 25, ClientRectangle.Height - 50 );
fEastLabel.Text = "0";
fEastLabel.TextAlign = ContentAlignment.MiddleCenter;
DrawingPanel Configuration

// place panel that hosts counter in the middle
fDrawingPanel.Dock = DockStyle.Fill;
fDrawingPanel.Location = new Point(25, 25);
fDrawingPanel.Size = new Size(ClientRectangle.Width - 50,
                                ClientRectangle.Height - 50);

// register event handlers
fDrawingPanel.Paint += // repaint counter
    new PaintEventHandler(HandlePaint);
fDrawingPanel.MouseDown += // start change critical section
    new MouseEventHandler(HandleMouseDown);
fDrawingPanel.MouseUp += // stop change critical section
    new MouseEventHandler(HandleMouseUp);
fDrawingPanel.MouseMove += // change critical section
    new MouseEventHandler(HandleMouseMove);
public event EventHandler EnterSection;
public event EventHandler LeaveSection;

private void DoEnterSection() {
    if (EnterSection != null) {
        EnterSection(this, new EventArgs());
        fInSection = true;
    }
}

private void DoLeaveSection() {
    if (LeaveSection != null) {
        LeaveSection(this, new EventArgs());
        fInSection = false;
    }
}
InCSec

\[ f\text{InSection} = \text{false}; \]

![Diagram showing states and transitions]

\[ f\text{InSection} = \text{true}; \]
.NET Geometry

\[
\begin{align*}
\text{X/Width} & \quad 180-270 \\
\text{Y/Height} & \quad 90-180 \quad 0-90 \\
\text{270-360} & \\
360 &= 2\pi r
\end{align*}
\]

\[
\begin{align*}
\alpha & \\
\sin \alpha & \\
\cos \alpha & \\
\tan \alpha & \\
\end{align*}
\]
private int GetSelectedAngle( Point aPoint )
{
    // X coordinate relative to the center of the panel
    int lXDelta = aPoint.X - (fDrawingPanel.ClientSize.Width / 2);
    // Y coordinate relative to the center of the panel
    int lYDelta = (fDrawingPanel.ClientSize.Height / 2) - aPoint.Y;
    // Atans returns the angle whose tangent is the
    // quotient of two specified numbers.
    double lSelAlpha = Math.Atan2((double)lYDelta, (double)lXDelta);
    // translate quadrants, .NET uses clockwise order
    return 360 - (((int)(lSelAlpha * 180.0D / Math.PI) + 360) % 360);
}
Selected Angle

\(X/\text{Width} = 100\)

\(Y/\text{Height} = 100\)

\((75,75)(75,25)\)

\((25,75)\)

\(\tan \alpha = \Delta Y / \Delta X = 1\)

\((25,75) = 45^\circ\), but in .NET \((25,75)\) is in the fourth quadrant

\(\Rightarrow 360 - 45 = 315^\circ\)
private void HandleMouseUp( object sender, MouseEventArgs e )
{
    // we only handle the left button
    if ( e.Button == MouseButtons.Left )
    {
        int lAngle = GetSelectedAngle( new Point( e.X, e.Y ) );
        lAngle = lAngle + STEP - (lAngle % STEP); // adjust angle
        if ( fStartSelected )
            SetStartSection( lAngle );
        if ( fStopSelected )
            SetStopSection( lAngle );
        Cursor = Cursors.Default;
    }
}
private void HandlePaint( object sender, PaintEventArgs e )
{
    Graphics lGraphics = e.Graphics;
    Rectangle lClientRec = fDrawingPanel.ClientRectangle;

    // draw actual arc
    lGraphics.FillPie( new SolidBrush( fArcColor ),
                       5, 5,
                       lClientRec.Width - 10, lClientRec.Height - 10,
                       fStart, fAngle );

    ...
}