

# Course IPCis: Image Processing with C#

## Chapter C1: Commented Code of the Bitmap Project

Copyright © by V. Miszalok, last update: 03-05-2007

---

```
using System; //Home of the base class of all classes "System.Object" and of all primitive data types such as Int32,
Int16, double, string.
```

---

```
using System.Drawing; //Home of the "Graphics" class and its drawing methods such as DrawString, DrawLine,
DrawRectangle, FillClosedCurve etc.
```

---

```
using System.Drawing.Imaging; //Home of the Bitmap-class.
```

---

```
using System.Windows.Forms; //Home of the "Form" class (base class of our main window Form1) and its method
Application.Run.
```

---

**Entry** `public class Form1 : Form`

//We derive our window `Form1` from the class `Form`, which the compiler automatically finds in the `System.Windows.Forms` namespace.

```
{ [STAThread] [STAThread] static void Main() { Application.Run( new Form1() ); } //Create a
single thread instance of Form1 and ask the operating system to start it as main window of our program.
```

---

```
Brush bbrush = SystemBrushes.ControlText; //Create a reference to an already existing Brush object (just a
programming shortcut).
```

---

```
Brush rbrush = new SolidBrush( Color.Red ); //Create a new global Brush object.
```

---

```
Bitmap bmp; //Global Bitmap-object
```

---

```
int nClicks; //Global counter of mouse clicks is used to switch through 8 different views of bmp.
```

---

**Constructor** `public Form1()`

```
{ MenuItem miRead = new MenuItem( "&Read", new EventHandler( MenuFileRead ) ); //Purpose: Open
an image file.
```

---

```
MenuItem miExit = new MenuItem( "&Exit", new EventHandler( MenuFileExit ) ); //Purpose: Leave the
program.
```

---

```
MenuItem miFile = new MenuItem( "&File", new MenuItem[] { miRead, miExit } ); //Complete menu
with two entries.
```

---

```
Menu = new System.Windows.Forms.MainMenu( new MenuItem[] { miFile } ); //Attach the menu to Form1.
```

---

```
Text = "Bitmap1"; //Blue title bar.
```

---

```
SetStyle( ControlStyles.ResizeRedraw, true ); //Redirect the OnResize-event to the OnPaint event handler.
```

---

```
Width = 1024; //Initial window size.
```

---

```
Height = 800; //Initial window size. Raise the OnResize-event which calls protected override void
OnPaint(...).
```

---

```
try { //Delete this and the following 6 lines if you have no Internet connection running.
//try-clause reads an image from the internet.
```

```
System.Net.WebRequest webreq = System.Net.WebRequest.Create(
```

```
"http://www.miszalok.de/Images/Madonna.bmp" ); //Try to reach a bitmap file.
```

```
System.Net.WebResponse webres = webreq.GetResponse(); //Wait for response.
```

```
System.IO.Stream stream = webres.GetResponseStream(); //Store the response if any.
```

```
bmp = (Bitmap)Image.FromStream( stream ); //Convert the response into a bitmap-object.
```

```
Invalidate(); //Call protected override void OnPaint(...).
```

```
} catch {}; //End of try-clause. If there was no response, forget about the WebRequest. In this case the user has to
load an image manually from his own hard disk.
```

```
Event Handler void MenuFileRead( object obj, EventArgs ea )
```

```
{ OpenFileDialog dlg = new OpenFileDialog(); //Call the standard modal OpenFileDialog-box.
if ( dlg.ShowDialog() != DialogResult.OK ) return; //Forget it, if there was no reasonable user reaction.
try { bmp = (Bitmap)Image.FromFile( dlg.FileName ); } catch { return; } //Try to read the image
using the powerful Bitmap-class. Forget it if that wasn't an image file.
nClicks = 0; //Initialize the mouse-click-counter.
Invalidate(); //Call protected override void OnPaint(...).
```

```
Event Handler void MenuFileExit( object obj, EventArgs ea )
```

```
{ Application.Exit(); } //Kill the current instance of program bitmap1.
```

```
Event Handler protected override void OnMouseDown( MouseEventArgs e )
```

```
{ nClicks++; //Increment the mouse-click-counter.
Invalidate(); //Call protected override void OnPaint(...).
```

```
Event Handler protected override void OnPaint( PaintEventArgs e )
```

```
{ Graphics g = e. Graphics; //Get a current Graphics-object in order to be able to draw something.
if ( bmp == null ) { g.DrawString( "Open an Image File !", Font, bbrush, 0, 0 ); return; }
//Inform the user what to do, if there is no image at all.
Rectangle cr = ClientRectangle; //Store the current size of the client area.
int line = 0; //Vertical position of the first printed line.
switch ( nClicks % 9 ) //Divide the mouse-click-counter by 9 and use the remainder as selector how to react. The
modulo operator furnishes cyclic results between 0 and 8 regardless of how many times a mouse button has been pressed.
{ case 0: //Information //The mouse button has been pressed 0, 9, 18 etc. times.
g.DrawString( "RawFormat = " + bmp.RawFormat.ToString(), Font, bbrush, 0, line+=Font.Height
); //Extract the RawFormat-property from class bmp. see http://msdn2.microsoft.com/en-us/library/system.drawing.image.rawformat.aspx.
if ( bmp.RawFormat.Guid == ImageFormat.Bmp.Guid ) //Extract the Guid-property from the RawFormat-
property and compare it with "Bmp.Guid". see http://msdn2.microsoft.com/en-us/library/system.drawing.imaging.imageformat\_properties.aspx.
g.DrawString( "BMP", Font, bbrush, 0, line+=Font.Height ); //RawFormat==Bmp.Guid means the
Windows-Bitmap Image File Format.
if ( bmp.RawFormat.Guid == ImageFormat.Jpeg.Guid ) //Extract the Guid-property from the RawFormat-
property and compare it with "Jpeg.Guid".
g.DrawString( "JPG", Font, bbrush, 0, line+=Font.Height ); //RawFormat==Jpeg.Guid means the JPG
Image File Format.
g.DrawString( "Width = " + bmp.Width.ToString() , Font, bbrush, 0, line+=Font.Height ); //Print
the no. of columns.
g.DrawString( "Height = " + bmp.Height.ToString() , Font, bbrush, 0, line+=Font.Height );
//Print the no. of rows.
g.DrawString( "PixelFormat = " + bmp.PixelFormat.ToString(), Font, bbrush, 0,
line+=Font.Height ); //Print the pixel format and no. of bits per pixel
g.DrawString( "Click on left mouse button !", Font, rbrush, 0, line+=Font.Height); //Advice the
user what to do.
break; //End of case 0: .
```

```

case 1: //Raw display //The mouse button has been pressed 1, 10, 19 etc. times.
-----
g.DrawImage( bmp, 0, Font.Height ); //Draw the original image in its native size below the existing line of text.
-----
g.DrawString( "Click on left mouse button !", Font, rbrush, 0, 0 ); //
-----
break; //End of case 1: .
-----

case 2: //Center //The mouse button has been pressed 2, 11, 20 etc. times.
-----
Int32 x = ( cr.Width - bmp.Width ) / 2; //Left margin.
-----
Int32 y = ( cr.Height - bmp.Height ) / 2; //Upper margin.
-----
g.DrawImage(bmp, x, y, bmp.Width, bmp.Height); //Center the original image in its native size in the middle of
the client area of Form1..
-----
g.DrawString( "Change window size ! Click on left mouse button !", Font, rbrush, 0, 0 );
//Advice the user what to do.
-----
break; //End of case 2: .
-----

case 3: //Horizontal stretch //The mouse button has been pressed 3, 12, 21 etc. times.
-----
x = 0; //Left margin = 0.
-----
y = ( cr.Height - bmp.Height / 2 ) / 2; //Upper margin.
-----
g.DrawImage( bmp, x, y, cr.Width, bmp.Height / 2 ); //full form width, half bmp height
//Stretch the image horizontally and squeeze it vertically.
-----
g.DrawString( "Change window size ! Click on left mouse button !", Font, rbrush, 0, 0 );
//Advice the user what to do.
-----
break; //End of case 3: .
-----

case 4: //Vertical stretch //The mouse button has been pressed 4, 13, 22 etc. times.
-----
x = ( cr.Width - bmp.Width / 2 ) / 2; //Left margin.
-----
y = 0; //Upper margin = 0.
-----
g.DrawImage( bmp, x, y, bmp.Width / 2, cr.Height ); //half bmp width, full form height
//Squeeze the image horizontally and stretch it vertically.
-----
g.DrawString( "Change window size ! Click on left mouse button !", Font, rbrush, 0, 0 );
//Advice the user what to do.
-----
break; //End of case 4: .
-----

case 5: //Full size //The mouse button has been pressed 5, 14, 23 etc. times.
-----
g.DrawImage( bmp, cr ); //Stretch the image in order to fit exactly into the client area of Form1.
-----
g.DrawString( "Change window size ! Click on left mouse button !", Font, rbrush, 0, 0 );
//Advice the user what to do.
-----
break; //End of case 5: .
-----

```

case 6: //Mirror//The mouse button has been pressed 6, 15, 24 etc. times.

---

```
g.DrawImage( bmp, cr.Width/2, cr.Height/2, cr.Width/2, cr.Height/2 ); //Upright into the lower right quadrant.
```

---

```
g.DrawImage( bmp, cr.Width/2, cr.Height/2, -cr.Width/2, cr.Height/2 ); //Horizontally mirrored into the left lower quadrant.
```

---

```
g.DrawImage( bmp, cr.Width/2, cr.Height/2, cr.Width/2, -cr.Height/2 ); //Vertically mirrored into the right upper quadrant.
```

---

```
g.DrawImage( bmp, cr.Width/2, cr.Height/2, -cr.Width/2, -cr.Height/2 ); //Horiz. and vert. mirrored into the left upper quadrant.
```

---

```
g.DrawString( "Change window size ! Click on left mouse button !", Font, rbrush, 0, 0 ); //Advice the user what to do.
```

---

```
break; //End of case 6: .
```

---

case 7: //Zoom animation//The mouse button has been pressed 7, 16, 25 etc. times.

---

```
x = cr.Width / 20; //5% of image width.
```

---

```
y = cr.Height / 20; //5% of image height.
```

---

```
for ( Int32 i = 0; i < 20; i++ ) //Draw 20 shrinking images.
```

---

```
g.DrawImage( bmp, 0, 0, cr.Width - x*i, cr.Height - y*i ); //Draw 20 images shrinking from full size to 5%.
```

---

```
g.DrawString( "Change window size ! Click on left mouse button !", Font, rbrush, 0, 0 ); //Advice the user what to do.
```

---

```
break; //End of case 7: .
```

---

case 8: //Rotation animation//The mouse button has been pressed 8, 17, 26 etc. times.

---

```
Single fx = cr.Width / 100; //Horizontal step width = 1% of image width.
```

---

```
Single fy = cr.Height / 100; //Vertical step height = 1% of image height.
```

---

```
PointF[] p = new PointF[3]; //Triangle with float coordinates
p[0].X=p[0].Y=p[1].X=p[1].Y=p[2].X=p[2].Y = 0. This triangle will rotate in 100 steps. p[0] starts as upper left corner (0,0), p[1] starts as upper right corner (cr.Width,0) and p[2] starts as lower left corner (cr.Width,cr.Height). Each step shifts p[0] to the right, p[1] to the bottom and p[2] upwards. Thus the three vertices will move along the top, right and left margin of the client area.
```

---

```
p[1].X = cr.Width; //Initialize p[1] to the upper right corner.
```

---

```
p[2].Y = cr.Height; //Initialize p[2] to the lower left corner.
```

---

```
do //The loop starts here.
```

---

```
{ p[0].X += fx; //Shift p[0] to the right.
```

---

```
p[1].Y += fy; //Shift p[1] to the bottom.
```

---

```
p[2].Y -= fy; //Shift p[2] to the top.
```

---

```
g.DrawImage( bmp, p ); //Draw the image that fits to the rotated triangle.
```

---

```
} while ( p[2].Y > 0 ); //Stop the loop when p[2] reaches the top margin of the client area.
```

---

```
g.DrawString( "Change window size ! Click on left mouse button !", Font, rbrush, 0, 0 ); //Advice the user what to do.
```

---

```
break; //End of case 8: .
```

---