Game Programming with C#.NET CF

Author: Mohit Chawla

Mindfire Solutions, www.mindfiresolutions.com

November 23, 2006

Abstract

This paper discusses some aspects of developing games for Windows Mobile/PocketPC devices using the .NET Compact Framework and C#.NET. Ten guidelines are presented for beginners in this area to learn and get going quickly!

• *Introduction*

The .Net Compact Framework (smart device development framework) is a subset of the .Net Framework targeting small devices such as Pocket PC and other Windows CE .Net devices. It brings the world of managed code to devices. Over the years, Compact Framework (CF) has emerged as a strong source of Application as well as Game development courtesy Microsoft.

Being a developer, even the very thought of devices excites me with its technology possibilities. I am sure you are also curious the same way. Compact Framework with its increased developer productivity and reduced time to market gives us the freedom to exploit all those attributes and dimensions of devices.

In this article I will discuss the key prerequisite for developing games targeting small devices and show how .Net Compact Framework handles them. In short, you will see how easy and fun it is to develop and optimize games using .Net Compact Framework.

Note:

This article assumes the reader possesses some knowledge of .Net Compact Framework and gaming basics. The article targets framework 1.1. Even if you have knowledge of C++ this article will not be alien to you.

• *Guidelines*

Often while developing a game for small devices there are requirements that need special consideration. Let's look at some of those common ones, in no particular order.

1. Full Screen Mode

Devices are available in numerous sizes and shapes. Most standard pocket pc's have resolution varying from 240X320 pixels, 240X240 to VGA devices with resolution of 480X640 pixels. Variety in the resolution of these devices forces us to think about the need of "real estate" on the device. Facilitating full screen mode will hide the taskbar as well as the Start icon we see at the title bar residing at the top-left of the device window. An application can set a form at full screen by changing windowstate to Maximized on form_load.

```
Form.WindowState = FormWindowState.Maximized;
```

If a menu bar is attached to the device then the above may not work. For such kind of scenario you will need an API call.

```
public const NET_FORM_CLASSNAME = "#NETCF_AGL_BASE_";
public const SHFS_HIDESTARTICON = &H20;

      [DllImport("aygshell.dll")]
      public static extern bool SHFullScreen(IntPtr hWnd,
      int nState);

      [DllImport("Coredll")]
      static extern IntPtr FindWindow(string lpClassName
      ,string lpWindowName);

      Calling the function:
      SHFullScreen(FindWindow(NET_FORM_CLASSNAME,
      this.Text), SHFS_HIDESTARTICON)
```

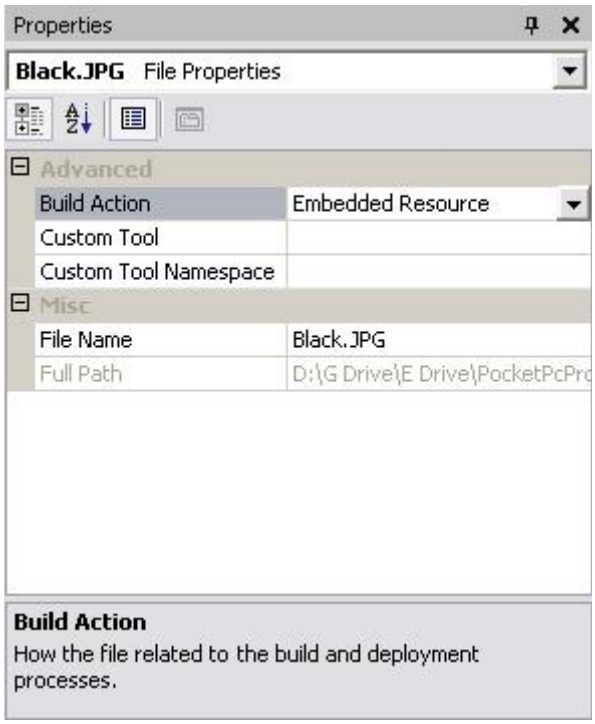
Generally, it is recommended not to change the client size as it might hinder the portability of your software to other pocket pc devices.

2. Image Build Action as Embedded Resource

When an assembly is created, we can store arbitrary files within it, such as XML's or BMP's etc. Those files are called Embedded Resource. To embed an image into an assembly, right click the image in Solution Explorer and click properties.
The key advantage of this action is that these "files" are then carried within the executable file. You don't have to worry about a user accidentally deleting or modifying the file, and thus breaking your code. Also embedding a resource within an assembly simplifies deployment.

Remember huge bitmaps should never be used as embedded resources as it increases the size of the executable.

The figure below shows the property window.



As shown above, change the Build Action of the image to Embedded Resource. To recover the images from an assembly use the following code.

```
Using System.Reflection;

Bitmap bmpImage = new Bitmap
(Assembly.GetExecutingAssembly().GetManifestResourceStream("Black.jpg")
);
```

The supported types for bitmap are JPG, BMP, GIF and PNG.

3. Bitmap Techniques for Painting

The architecture of graphical output involves a Shared managed code library "system.Drawing.dll". Windows CE supports a selected set of GDI drawing functions. There is no library explicitly named GDI in Windows CE, unlike Windows Desktop edition. Instead, the graphical functions reside in the coredll.dll library. These functions are exactly like their desktop counterparts, so even if there is no library named GDI in Windows CE, we refer to these functions as GDI functions.

The following information shows the namespaces supported by .Net CF.

- System.Drawing: Supported in .Net CF. A minimal set that allows for drawing of text, raster and vector objects with no built–in coordinate transformation
- System.Drawing.Drawing2D: Not supported in .Net CF (except for the CombineMode enumeration)
- System.Drawng.Imaging: Not supported in .Net CF (except for the ImageAttributes class)
- System.Drawing.Printing: Not supported in .Net CF
- System.Drawing.text: Not supported in .Net CF

The Graphics Class

Graphics is the class that supports graphics manipulations. Infact, this is the most important class for graphical outputs. This class holds methods such as DrawString, DrawImage, DrawRectangle, DrawLine, MeasureString etc.

Following table summarizes methods for text drawing supported by System.Drawing.Graphics.

Method	Description
MeasureString	Measures a specified string when drawn with a specified font. e.g. Graphics graphic = this.CreateGraphics(); System.Drawing.Font fntString = new Font("verdana",8,FontStyle.Regular); System.Drawing.SizeF fontsize = graphic.MeasureString(“String Calculation”,fntString);
DrawString	Draws the line of text for a specified font and text color. e.g. graphic.DrawString (“Text Print”, new Font("verdana",9,FontStyle.Regular),new SolidBrush(Color.Black), 0, 0);

Following are some commonly used System.Drawing.Graphics methods for Raster Drawing.

Method	Description
DrawImage	Draws the image in the display screen or wherever specified. e.g. Bitmap Animate = New Bitmap("\program files\Hollywood\Animate.JPG"); graphic.DrawImage (Animate, 0, 0, new Rectangle (0, 0, this. Animate.Width, this.Animate.Height), GraphicsUnit.Pixel);

Clear	Clears the entire drawing surface and fills it with the specified background color.
	e.g.
	e.Graphics.Clear(Color.Red);
	or
	Graphics graphic = this.CreateGraphics();
	graphic.Clear(Color.Teal);

Following are some commonly used System.Drawing.Graphics Methods for Vector Drawing.

Method	Description
DrawEllipse	Draws an ellipse with a pen. e.g. Pen redpen = new Pen(Color.Red); graphic.DrawEllipse(redpen, 0, 20, 10, 10);
DrawLine	Draws a straight line with a pen e.g. graphic.DrawLine(redpen, 0, 20, 20, 0);
DrawRectangle	Draws a rectangle specified by a coordinate positions, a width, and a height. e.g. graphic.DrawRectangle (new Pen(Color.Black),8, 10, this.Width, this.Height);
FillRectangle	Fills the interior of a rectangle with brush. e.g. SolidBrush brush = new SolidBrush(Color.FromArgb(0,255,0)); graphics.FillRectangle (brush, X, Y, pWidth, pHeight);

It is always crucial to dispose off the on screen graphics objects once they are no longer needed. Failure in doing so may result in deficit of resources for display.

Coming back to graphics, there are 2 ways to retrieve the graphics object. One of which is through the OnPaint () method using PaintEventArgs.Graphics. These graphics objects will automatically get disposed as soon as it moves out of method scope. While the second way, (as shown in an example earlier) is by making use of this.CreateGraphics () which returns a graphics object.

4. Gameboard

The flow of a game is controlled by game board or game controls. Therefore on a device the navigation controls (namely up, down, right, left) play a vital role in games. Below is an example which takes input from user and provides direction to the image.

```
private void hollywood_KeyDown(object sender,
System.Windows.Forms.KeyEventArgs e)
{
   if (e.KeyCode == Keys.Right || e.KeyCode == Keys.Left ||
e.KeyCode
        == Keys.Up || e.KeyCode == Keys.Down)
   {
      switch(e.KeyCode)
      {
         case Keys.Right:
         {
            _hollywood.MoveDirection = MoveDirections.Right;
            break;
         }
         case Keys.Left:
         {
            _hollywood.MoveDirection = MoveDirections.Left;
            break;
         }
         case Keys.Down:
         {
            _hollywood.MoveDirection = MoveDirections.Down;
            break;
         }
         case Keys.Up:
         {
            _hollywood.MoveDirection = MoveDirections.Up;
            break;
         }
      }
   }
}
```

5. Timer on the run

A game normally uses splash screens to show information relevant to that game. A splash screen is a window that appears for a specified time and vanishes as soon as the time is elapsed. To support such kind of scenario we make use of a timer control. The interval property of this control depicts the time interval in which the timer's tick event should be called.

Also to move objects that do not need user input we can make use of a timer control. Consider the example where 5 frames have to be put on top of the other to give an impression of a moving object. Placing them in a timer_tick event and setting the timer's interval to 1000 i.e. 5 frames per second (fps), will cause a moving effect on the screen.

6. Refresh and Invalidate

To progress the flow of a game, we have to refresh the screen or a part of it. Many times the paint method needs to be called explicitly, in order to achieve this we call this.Invalidate (). To invalidate /refresh only a selected area we can use the following code.

```
Rectangle rect = new Rectangle (0, 0, pnlInvalid.Width,
     pnlInvalid.Height);

this.Invalidate(rect);
```

Some times Invalidate alone doesn't call paint, for which we have to call this.Update () followed by Invalidate function.

The this.Refresh () function is also available but will always paint the whole screen.

7. Garbage Collection

The memory of a device is limited. It is therefore recommended to dispose memory as soon as it is no longer required. Failure in disposing the objects no longer used will result in an OutOfMemoryException stating "There is not enough memory available to

complete the operation". Components such as timer, imagelist, arrays while programming a game should be disposed as soon as it goes out of score.

Even a small hint of memory leak in your application may cause it to crash.

8. Making things move with Sprites

In order to create animation, an illusion of movement is created by displaying a rapid succession of images with slender changes in their appearance. More specifically, the human eye can be tricked into perceiving animated movement by displaying a sequence of pre-generated, static frame images (frame based animation). Cast based animation commonly referred to as sprites animation, involves graphical objects that move independent of a background.

Just like windows forms, sprites are placed at x, y positions. 0, 0 is the top, left position on a form whereas ClientSize.Width, ClientSize.Height are the bottom, right of the form. An increment in the number of sprites smoothens the final visual effect of animation.

The paragraph above gives you an understanding of the basic types of sprites including cast based animation used in animation to make things move. Following is a list the reveals the properties of a sprite that must be accounted for the Sprite Class:

- Position
- Velocity
- Z-order
- Bounding rectangle
- Bounds action
- Hidden/Visible

The most important property of a sprite is its position on the game board, followed by its velocity. The velocity of the sprite can be used to change the position of the sprite throughout the game. So if a sprite has an X velocity of 1 and Y velocity of -5, then it will move 1 pixel right and 5 pixels down the game board in every game cycle. In addition to it, it is always helpful to set the Z-order to every sprite with respect to the screen. Let me explain, if two sprites are placed on the same position on the screen, the one with the higher Z-order will be on top of the other. A z-order is an easier property to maintain.

A bounding rectangle on the other hand is very useful i.e. a rectangle that determines in which direction will sprites travel. For example, if a horse is a sprite and a pond be a bounding rectangle. Now if we want to show a horse moving around that pond, this can be easily accomplished by setting the bounding rectangle for the horse to be a rectangle that encompasses the pond.

Using a Bounding Rectangle, a bounding action is a way to determine how a sprite acts when it encounters a boundary.

The last property is worth considering in a sprite class is the sprite's visibility. There are many situations where we need to hide a sprite and not delete it. For example: you may have fishes in a pond which you wish to trap. SO to create that effect of a fish going into water and coming back can be done by hiding the fishes and then showing again until they are caught.

Example:

Sprite is at (3, 10) and has a speed of (2, 0). (2,0) means the sprite will move right in the x direction only. (3,10) + (2,0) = (5,10). If you repeat this addition every second, then the sprite will move 1 pixel per second across the screen.

9. Furnishing Transparency with Objects

A very crucial concern regarding sprites is transparency. Bitmap images are rectangular by nature; a problem arises when sprite images are not rectangular. In such scenario, the pixel surrounding the sprite image is unused. That is, when pixels of the unused area are encountered by drawing routines, they are simply skipped to give a smooth effect to the image leaving the original background intact.

10. Collision Detection

No discussion about games and animations would be complete without addressing Collision Detection (CD). CD is a method to ascertain whether sprites have collided with each other. Although this technique seems easy or you might think "what's the big deal" in detecting such a scenario. All we have to do is check for coordinates if they at all converge, but consider the condition where we have many sprites moving and each sprite has to be compared with each and every other sprite moving along. Accuracy of positioning is very critical in this case. Such situations are more or less difficult to manage. Almost all arcade games have to undertake collision detection for better results.

Example:
If a bullet sprite collides with human sprite, an explosion should occur. But if collision detection is not optimum the explosion time will differ from intersection point.

```
public bool DetectCollision()

{
    return! ( _Bulletsprite.Location.X > _Humansprite.Location.X +
      _Humansprite.Size.Width
      || _Bulletsprite.Location.X + _Bulletsprite.Size.Width  <
      _Humansprite.Location.X
      || _Bulletsprite.Location.Y > _Humansprite.Location.Y +
      _Humansprite.Size.Height
      || _Bulletsprite.Location.Y + _Bulletsprite.Size.Height <
      _Humansprite.Location.Y);
}
```

Collision detection becomes more complicated as you include irregular shapes, and if you are dealing with 3-dimensional objects.

• *Conclusion*

If you are working on Windows Mobile 5.0, you may come across many new features. Windows Mobile 5.0 has introduced operating system features directly to the developers. Some of those new native API's are:

- Direct3D Mobile: Developers can take advantage of the existing computer Direct3D skills. Direct3D takes advantage of graphics hardware support.
- DirectDraw: Applications may need to manipulate display memory directly and provide high performance 2-D graphics. Windows Mobile provides DirectDraw.
- DirectShow: Applications that want to interoperate with the camera, that is provided inbuilt with the device can take full advantage of the DirectShow API.
- Global Positioning Systems (GPS) APIs

• *Further Reading*

Though sound and music is a very critical feature of any game, I have kept it out of scope of above discussion. Similarly, I have tried to bring out some concepts which will be useful to you, but have no means covered all topics required to make an actual Pocket PC game!

Mindfire Solutions is an offshore software services company in India. Mindfire possesses expertise in multiple platforms, and has built a strong track record of delivery. Our continued focus on fundamental strengths in computer science has led to a unique technology-led position in software services.

If you want to learn more about us and our capabilities, please drop us an email at info@mindfiresolutions.com. We will be glad to help you.

To know more about Mindfire Solutions, please visit us on www.mindfiresolutions.com

---