

Timers, Animation, Images, Bitmaps

Windows Timer

- ⌘ Input device that periodically notifies an app each time a specified time interval has elapsed
- ⌘ Using a timer guarantees that a program can regain control periodically
- ⌘ Three different Timer classes in:
 - System.Timers, System.Threading, System.Windows.Forms
- ⌘ We'll use the last one – The same one that is available in Win32 API and MFC
 - It's integrated with other Windows events and is easiest to use

Timer applications

- ⌘ Implementing a clock
- ⌘ Multitasking
- ⌘ Maintaining updated status report
- ⌘ Implementing autosave feature
- ⌘ Terminating demo versions of programs
- ⌘ Activation of a screen saver after certain time
- ⌘ Pacing movement – animation
- ⌘ Others

The Timer Class

- ⌘ Creating a Timer object:
 - Timer timer = new Timer();
- ⌘ Timer class has one event:
 - Event: Tick
 - Delegate: EventHandler
 - Defining the Timer Tick event handler:
Void TimerOnTick(object obj, EventArgs ea) {...};
 - Attaching it to the Tick event:
timer.Tick += new EventHandler(TimerOnTick);
- ⌘ Timer read/write Properties:
 - int Interval, Tick time in milliseconds
 - bool Enabled, True if timer is running
- ⌘ Timer Methods:
 - void Start(); void Stop();

Some Timer Examples

- ⌘ CloseInTen:
 - A program that sets a “one-shot” timer that closes the application after ten seconds
 - Could be used to implement a “demo” version of a program that allows the user to try it for a while
 - Note use of obj argument in TimerOnTick() handler to get the timer that sent the message
- ⌘ RandomRectangles-timer:
 - Draws a new random rectangle once every 2 seconds
 - ⌘ We must use CreateGraphics() to create a Graphics object to draw with
- ⌘ Note that a timer can be programmed manually...
- ⌘ Or by using the Designer
 - Just drag a timer into the Form and double click on it to add the Timer Tick event handler
 - Set the Enabled and Interval properties in the Properties window

Animated Graphics

- ⌘ Creating a moving picture
 - Give illusion of motion by continual draw/erase/redraw
 - If done fast, eye perceives moving image
- ⌘ In a single-user (DOS) application, we could do the following:

```
Do Forever{  
    /* compute new location of object */  
    /* erase old object image */  
    /* draw object at new location */ }  
}
```

- ⚡ In Windows, other programs can't run while this loop is executing
- ⚡ Need to keep giving control back to Windows so other programs can operate
- ⚡ Ways of doing it:
 - Use PeekMessage() Loop (for Win32 API)
 - Override OnIdle() (for MFC)
 - Use a Windows Timer (any Windows platform)
 - ⚡ Erase old frame and draw new frame each time there is a timer 'tick' event

Bouncing Ball Example Program

- ⚡ Draws a red ball moving inside window's client area at a given velocity
- ⚡ Responds to form's Resize event to reset ball's position when window is resized
- ⚡ Responds to Timer event to draw next animation frame
- ⚡ Class level variables (accessible to all class methods):
 - xC, yC: current coordinates of ball's center
 - cxmove, cymove: x,y components of velocity
 - iXSize, iYSize: dimensions of window's client area
- ⚡ Helper function DrawBall()
 - Uses the Form's CreateGraphics() method to get a Graphics object
 - Draws BackColor ellipse in old position and red one in new posn.
 - ⚡ After each timer tick and after window is resized
 - Checks for collisions with sides of window and adjusts ball's path

DateTime Structure in .NET

- ⚡ To keep track of time and date
- ⚡ Some Constructors:
 - DateTime(int year, int month, int day);
 - DateTime(int year, int month, int day, int hour, int minute, int second);
 - DateTime(int year, int month, int day, int hour, int minute, int second, int msec);
- ⚡ year: 1-9999, month: 1-12, day: 1-#days in month, hour: 0-23, minute: 0-59, second: 0-59, msec: 0-999

DateTime Properties

- ⚡ Some Read-only Properties
 - Year, Month, Day, Hour, Minute, Second, Millisecond, DayOfWeek, DayOfYear
- ⚡ An important Static Property
 - Now
 - ⚡ Returns a DateTime structure filled with current local date and time
 - ⚡ E.g., to get current date and time:


```
DateTime dt = DateTime.Now;
```

 - dt then contains the current date/time

Some DateTime Methods

- string ToString()
 - ⚡ dt.ToString();
 - ⚡ Returns something like: "10/1/2004 10:30:01 A.M."
- string ToString(string strFormat)
 - ⚡ strFormat and returned values:
 - "d" 10/1/2004
 - "D" Friday, October 01, 2004
 - "f" Friday, October 01, 2004 10:30 A.M.
 - "F" Friday, October 01, 2004 10:30:01 A.M.
 - "g" 10/1/2004
 - "G" 10/1/2004 10:30:01 A.M.
 - "m" October 1
 - "t" 10:30 A.M.
 - "u" 2004-10-01 10:30:01

A Simple Digital Clock Program (SimpleClock)

- Uses a one-second timer
- Each timer tick the handler calls Invalidate() to force a Paint message
- Paint handler calls DateTime.Now Property to get a DateTime object containing the current time and date
 - ⚡ The DateTime object's ToString() method converts it to the appropriate string format
 - ⚡ DrawString() draws the string at the top of the Form's client area

Images and Bitmaps

- ✎ Video display images described by Images and/or Bitmaps
 - Rectangular arrays of "pixel values" stored in memory
 - Pixel value determines color of a pixel
 - Encapsulated in .NET Image and Bitmap classes
- ✎ Can be created and edited with almost any paint program
- ✎ Windows supports 4-bit, 8-bit (indirect) and 16 or 24-bit (direct) pixel values
- ✎ Can be stored/retrieved as .bmp files
 - Take up lots of space (no compression)
- ✎ Other common file formats (some compressed):
 - Jpg, Gif, Png, Tiff

- ✎ Can be displayed on a device using `DrawImage()` method of the `Graphics` object (`gr-obj`) associated with a device, e.g.:
 - `gr-obj.DrawImage(Image img, int x, int y);`
 - `gr-obj.DrawImage(Image img, point pt);`
 - Lots of other overloads available (as we'll see)
- ✎ Can be manipulated invisibly and apart from physical display device
- ✎ Fast transfer to/from physical device ==> flicker free animation
- ✎ Does not store information on drawing commands
 - Windows Metafiles do that
- ✎ You can also draw on an `Image` or `Bitmap`
 - Then transfer it to the screen
 - One screen access ==> no flicker in animations

System.Drawing.Image Class

- ✎ An abstract class
 - Can't be instantiated with a constructor
 - But has overloaded static methods that return `Image` objects that can be displayed
 - Can load a bitmap or metafile from a file
 - `Image img = Image.FromFile(strFilename);`
 - `Bitmap bmp = (Bitmap)Image.FromFile(strFilename);`
 - ✎ Other overloads
 - Once you've loaded an `Image`, you can use `DrawImage(img, ...)` to display it

Two Example Programs

- `ImgFromFile`
 - ✎ Displays a jpg image on the window's client area
 - But what if image file is not in right directory?
 - `FromFile()` method will throw an exception and program will die
 - Our program should be able to catch that exception
 - ✎ And do we need to retrieve the image - i.e. call `FromFile()` -- every time there's a `Paint` event?
- `ImgFromFileBetter`
 - ✎ Uses a try/catch block to avoid errors
 - Puts up a `MessageBox` if there is an exception
 - ✎ And makes the call to `FromFile()` in program's constructor
 - Stores the `Image` in a class level variable so it's accessible to the `Paint` handler

Other Image Class & Image Drawing Information

- ✎ Some `Image` Properties (read-only):
 - `Size`
 - ✎ Represents the size of the rectangular image (`Width` & `Height` members)
 - Members: `int Width, int Height`
 - ✎ Width and height of the image in pixels
- ✎ Other overloads of `DrawImage()` that specify a rectangular destination and/or source region for the image:
 - `DrawImage(Image img, int x, int y, int w, int h);`
 - ✎ `x, y` = position; `w` = width, `h` = height of image on destination window
 - `DrawImage(Image img, Rectangle rectDst);`
 - ✎ `rectDst` specifies rectangle on window image will be displayed in
 - Some read/write properties of `Rectangle` class:
 - `X, Y` Coordinates of upper left hand corner
 - `Width, Height`
 - `DrawImage(Image img, Rectangle rectDst, Rectangle rectSrc, GraphicsUnit gu);`
 - Arguments:
 - ✎ Destination and source `Rectangles`
 - ✎ `GraphicsUnit` enumeration value must be `GraphicsUnit.Pixel`
 - With these we can stretch or compress all or part of an image

More Image Examples

- ✎ `ImgCenter`
 - Maintains image in center of window's client area
- ✎ `ImgScaleToWindow`
 - Scales image to fit in window's client area
- ✎ `ImgPart`
 - Displays part of image
- ✎ `ImgPartScale`
 - Scales part of image to fit in window's client area

Rotating & Shearing an Image

`DrawImage(Image img, Point[] apt);`

- apt is an array of three points:
 - apt[0] = destination of upper left corner of image
 - apt[1] = destination of upper right corner of image
 - apt[2] = destination of lower left corner of image
 - 4th point generated automatically completes parallelogram

`DrawImage(Image img, Point[] aptDst, Rectangle rectSrc, GraphicsUnit gu);`

- aptDst: an array of three points specifying three corners of the image (as in `DrawImage`)
- rectSrc: source rectangle of original image
- gu: Source rectangle unit of measure enumeration value
 - Display, Inch, Millimeter, Pixel, Point, etc.
- Depending on the points in the array, the image will be rotated and/or sheared
- Example Program: `ImgAtPoints`

Drawing on an Image

- Up to now we've drawn an image on a Graphics object
 - Refers to the video display
 - The GDI+ is really drawing on a huge bitmap stored in memory
 - Refers to the screen's video display adapter
- But we can draw on any bitmap
 - First must get a Graphics object that refers to the image
 - Use `Graphics.FromImage(Image img)` static method to get it:
 - `Graphics g = Graphics.FromImage(img);`
 - Draw on it with GDI+ drawing functions
 - Display it by getting a screen Graphics object and using one of its `DrawImage(img, ...)` methods
 - Done typically in Paint handler
 - Must Dispose of image's graphics object after using it
 - `g.Dispose();`
- Example: `ImgDrawOn`

"Shadow" Images

- We may want to compose a complex scene off screen - a "shadow bitmap" or "shadow image"
 - Draw on a graphics object that refers to the shadow image as much as you like outside of Paint handler so you're not accessing the physical screen
 - Even draw other images on the shadow image!
 - Then in Paint handler (or in response to timer ticks), display it with a single call to `DrawImage(bitmap, ...)`
 - See `ImgShadowBitmap` example
- Very useful in avoiding flicker in animations
 - "Compose" the next frame in the shadow image
 - Draw all the objects on it first
 - Then draw the "composed" image on the physical screen
 - Thus only one access per frame to the physical screen
 - This technique is called "double buffering"

Bitmap Class

- Like Image class, but you can do more with it
- Create a blank bitmap of a specified size with constructor:
 - `Bitmap b m = new Bitmap(int width, int height);`
- Used like Image objects in drawing pictures and in double buffering
- Nice for making parts of a sprite "transparent"
 - So there is no rectangular "halo" around the sprite when it is drawn over the background
 - For example for a sprite that has a white background:
 - `Bitmap sprite = (Bitmap)Image.FromFile(sprite-file.bmp);`
 - `sprite.MakeTransparent(Color.White);`
 - Then draw as usual onto a shadow bitmap's graphics object
 - See `ImgShadowBitmap2` example