.NET Graphics Units and Transformations

The Windows Clipboard

Device Independent Drawing and Changing Coordinate System

- Want to be able to control how the output of our programs looks on different devices
  - Printers and video devices use different units
- Also want to be able to use different coordinate systems
  - Different units (inches, centimeters, device units)
  - Change direction of coordinate axes
  - Rotate, translate, scale what is displayed

Video and Printer Coordinates

- Device coordinates: x-axis to the right, y-axis down
- Video uses pixel units
  - –Resolution of most video display modes is about 100 dpi (dots per inch)
- For printer, printer dot units, but under the GDI+ …
  - –Coordinates passed to its Graphics drawing functions are interpreted as 0.01 inch units, regardless of printer
- So for most applications we'll get about the same results when we use the same coordinates to draw on a video Graphics object and a printer Graphics object

Manual Coordinate System Conversions

- Use the DpiX and DpiY properties of the Graphics object to adjust coordinates passed to drawing functions
  - –Horizontal and vertical resolution of device in dots per inch
- Example: want to use floating-point coordinates to draw in millimeter units
  - –A helper function to convert a point from millimeters to dots (pixels or printer dots):

```
PointF MMConv(Graphics g, PointF pointf)
// returns a PointF object which is the conversion of pointf parameter to dots
{
    pointf.X *= g.DpiX/25.4f;  // divide by 25.4 mm/inch to convert to inches
    pointf.Y *= g.DpiY/25.4f;  // and multiply by Dpi to convert to dots
    Return pointf;
}
```

Page Units & Page Scale

•Graphics class has two properties that do unit conversions (scalings) for us, so we don't have to write our own function to do them:

GraphicsUnit PageUnit // units used, an enumeration

- •Members
  - –Display Same as device (pixels or printer dots)
  - –Pixel
  - –Point units of 1/72 inch
  - –Inch
  - –Document units of 1/300 inch
  - –Millimeter

float PageScale // how much to scale by

The Page Transformation

•Example: we want to draw in units of hundredths of an inch

```
g.PageUnit = GraphicsUnit.Inch;
g.PageScale = 0.01f;
```

•Then to draw a 1-inch long horizontal line:

```
g.DrawLine(pen, 0, 0, 100, 0);
```

- •Resulting line on a printer would be exactly 1 inch long

•Setting PageUnit and PageScale properties is known as the Page Transformation

- –GDI+ applies two formulas to get device coordinates from page coordinates sent to Graphics functions

```
xdevice = xpage * PageScale * xconvfactor
ydevice = ypage * PageScale * yconvfactor
```

- –Conversion factors depend on PageUnit
  - »Display (printer): xconvfactor = DpiX/100; yconvfactor = DpiY/100
  - »Display (video): xconvfactor = yconvfactor = 1
  - »Inch: xconvfactor = DpiX, yconvfactor = DpiY
  - »Millimeter: xconvfactor = DpiX/25.4; yconvfactor = DpiY/25.4

Ten-Centimeter-Ruler Example Program

- Draws a ruler that's exactly 10 centimeters long with mm., half-cm. markings and numbers
- The Form's class is subclassed from our PrintableForm class ("enhanced") so that we can override its DoPage() method to draw on either the window's client area or to the default printer (if user selects "Print" menu item)
  - The "enhanced" PrintableForm class has a "Print Preview" menu item whose handler starts a PrintPreview dialog box to view a preview of the printer output

Client Area Dimensions

- ClientSize property always gives the dimensions of the client area in pixels
- After setting a new page transformation, you may need the dimensions in the units being used to draw with
  - Use VisibleClipBounds property of the graphics object
    - For video display it returns the dimensions of the client area in whatever page units you are using
    - But for printer, VisibleClipBounds returns printable area in units of 1/100 inch
      - So for printer you may need to convert the value returned by ClientSize to Page Units

Transforming between Page Units and Device Units

g.TransformPoints(CoordinateSpace.Page, CoordinateSpace.Device, apt);

- –To convert points from device units to page units
- –Converts an array of points, apt, from device to page coordinates
  - »apt is the original array of points
  - »Result will be in apt

g.TransformPoints(CoordinateSpace.Device, CoordinateSpace.Page, apt);

- –Converts an array of points, apt, from page units to device units. Result will be in apt

World Coordinate Transformation

- Page Transformation limitations:
  - –Must have same units in horizontal and vertical directions (no non-isotropic scaling)
  - –x-axis must point right and y-axis down
  - –Origin must be at (0,0)
  - –Cannot do rotations or translations
- To overcome these and other limitations, use a World Coordinate Transformation
  - –GDI+ uses 3 X 3 homogeneous transformation matrices to map world coordinates to device coordinates

Graphics Class Methods that Implement the World Coordinate Transformation

- RotateTransform(angle-in-degrees)
  - –Clockwise is positive
- ScaleTransform(sx, sy)
  - –sx, sy are horizontal and vertical scaling factors
- TranslateTransform(tx, ty)
  - –Shifts x and y coordinates along horizontal and vertical axes
- All subsequent Graphics drawing functions will use these new coordinates
- See World-Transformation example program
  - –Also uses "enhanced" PrintableForm class

The Windows Clipboard

Transferring Data Between Applications with the Clipboard

- The Windows Clipboard
  - –A global memory block maintained by the Windows OS
  - –Available since the earliest versions of Windows
  - –Provides a common mechanism for getting data from application to application
    - Also be used for single-program cut/copy and paste
  - –Many programs have "Cut", "Copy", "Paste" menu items
    - "Cut" or "Copy": transfers data to clipboard
    - "Paste": retrieves data from clipboard
  - –Clipboard contents are available to any running app

The Clipboard Class

- It's in System.Windows.Forms namespace
- It's tiny, but powerful
  - –No properties and just two static methods:
    1. void SetDataObject(object obj) or
       void SetDataObject(object obj, bool bStayAfterAppExit)
       - –Copies obj to the clipboard memory block
       - –Object could be many data types:
         - »String, Bitmap, etc.
       - –Two successive calls replace first object with second on clipboard
       - –If 2nd argument is false, item on clipboard is gone after program terminates
    2. IDataObject GetDataObject()
       - –Returns an instance of a class that implements IDataObject "interface"
         - »Interface: like a class; only contains method signatures, no bodies
         - »If a class inherits from an interface it must implement all methods

Using IDataObject

IDataObject data = Clipboard.GetDataObject();

–Has two methods: GetDataPresent(), GetData()

data.GetDataPresent(DataFormats format);

- •Returns true if the object retrieved contains data of the specified data format
- •Some DataFormats:
  - –Bitmap, Dib, Html, RTF, Serializable, StringFormat, Text, Tiff, WaveAudio, … lots more

–Some examples:

```
if (data.GetDataPresent(typeof(string))) {…}
if (data.GetDataPresent(DataFormats.Bitmap)) {…}
```

- •After verifying that the desired data type is on the clipboard, retrieve data with its GetData() method, e.g.,

```
string str = (string)data.GetData(typeof(string));
Bitmap bm = (Bitmap)data.GetData(DataFormats.Bitmap);
```

Clipboard-Simple Example

- •Form has a multi-line text box and a picture box
  - –PictureBox control class can display an image or bitmap
    - •Set SizeMode property to "StretchImage"
      - –So any image displayed will be scaled to fill the picture box
- •"Edit" menu item has "Copy" and "Paste" items
  - –If user clicks "Copy", selected text in textbox is sent to the clipboard
  - –If user clicks "paste"
    - •Any text on the clipboard is displayed in the textbox
    - •Any image on the clipboard is displayed in the picture box