Sketcher (Part 1)

Building Graphical User Interfaces (GUIs)

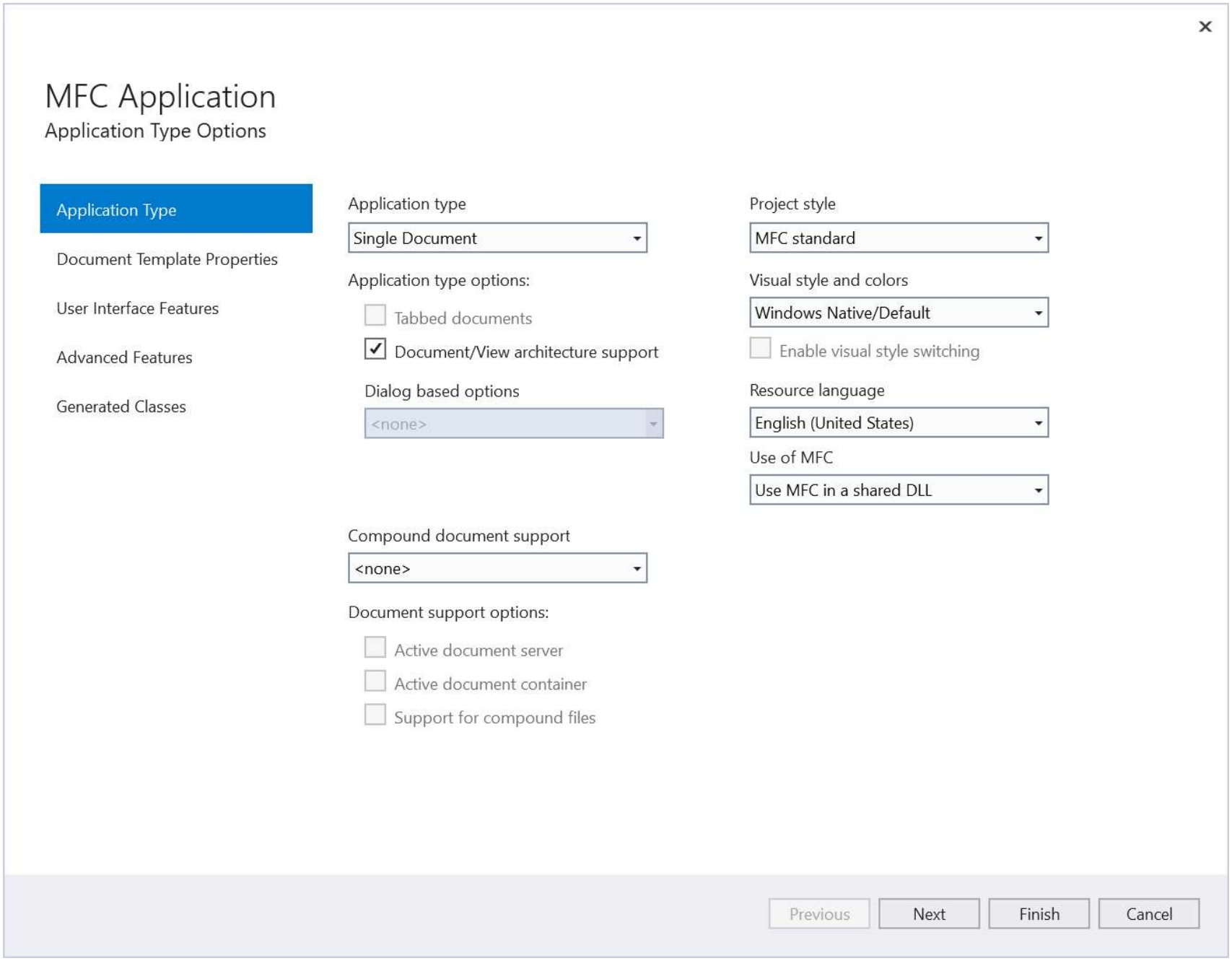
Using the Microsoft Foundation Classes (MFC)

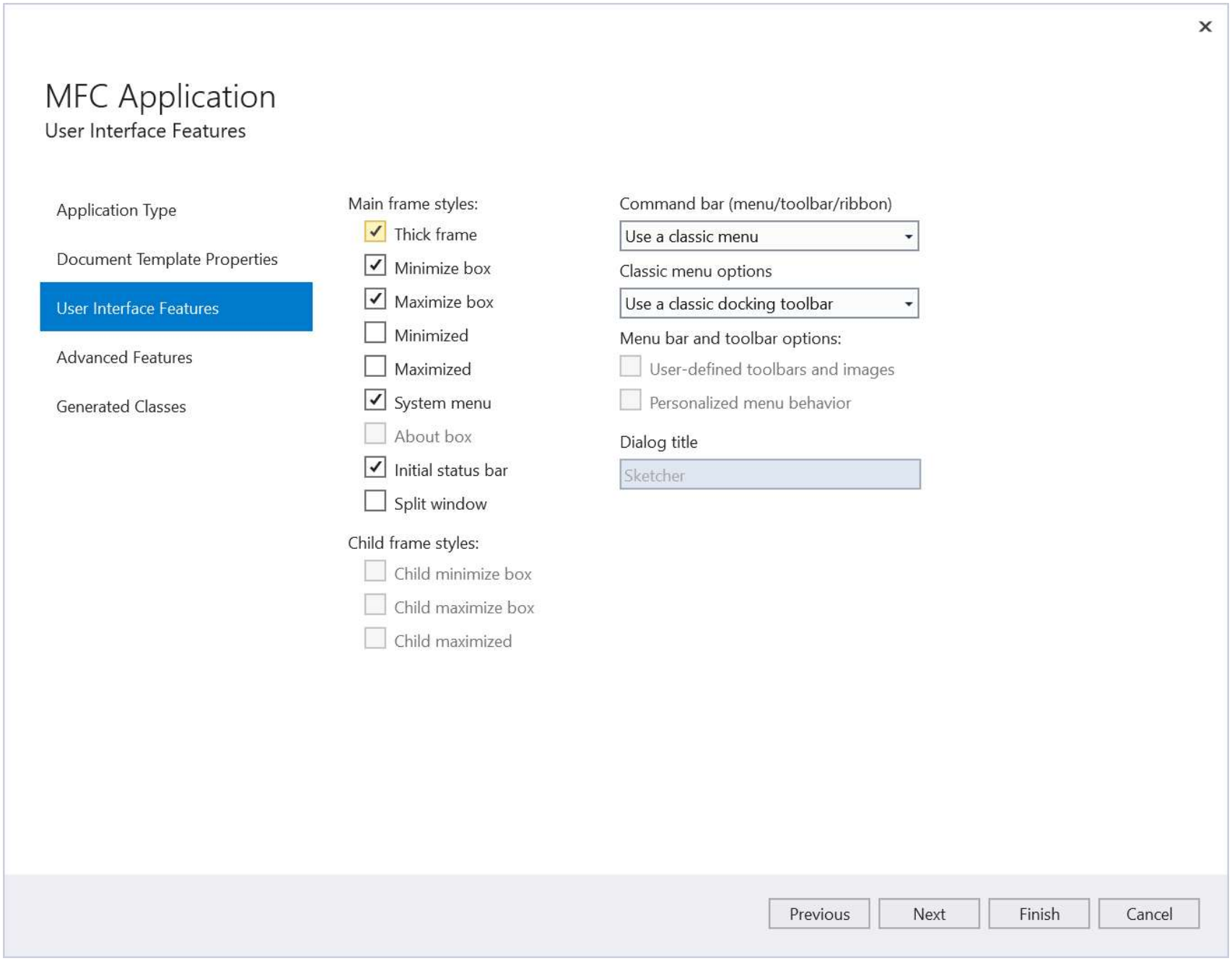
All of these examples assume that Microsoft Visual C++ 2017 is the compiler being used. Based on a tutorial written by Dr. Rick Coleman.

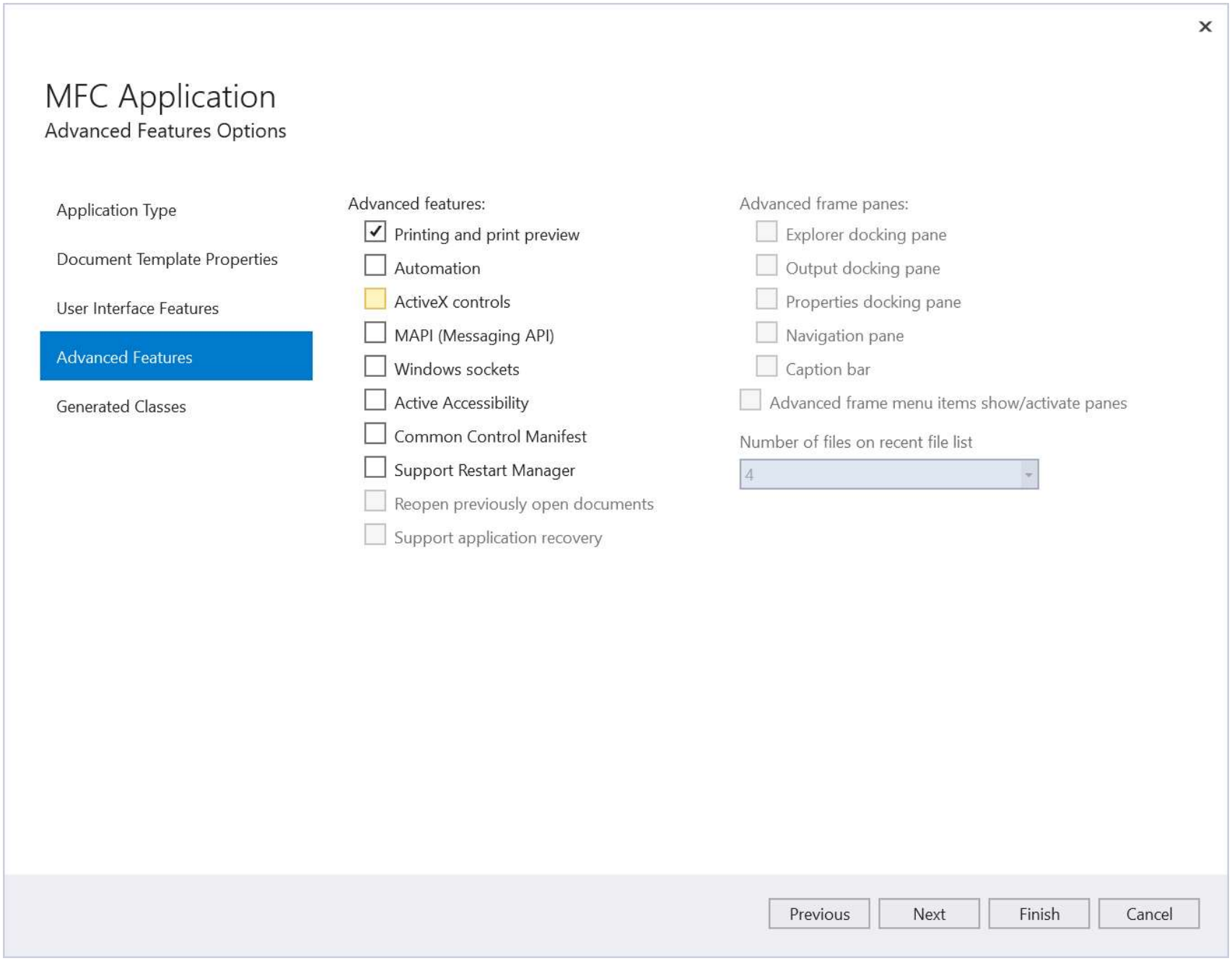
You also must have the "Visual C++ MFC for x86 and x64" feature enabled in the VS Installer.

Creating the project

Create a new MFC project (the same way we did before) with whatever name you want (preferably "Sketcher") and configure it according to the screenshots shown below. Additionally, change the "Character Set" for the project to "Not Set."


MFC Application
Application Type Options
Application Type
Document Template Properties
User Interface Features
Advanced Features
Generated Classes
Application type
Single Document
Application type options:
Tabbed documents
Document/View architecture support
Dialog based options
<none>
Compound document support
<none>
Document support options:
Active document server
Active document container
Support for compound files
Project style
MFC standard
Visual style and colors
Windows Native/Default
Enable visual style switching
Resource language
English (United States)
Use of MFC
Use MFC in a shared DLL
Previous
Next
Finish
Cancel


MFC Application
User Interface Features
Application Type
Document Template Properties
User Interface Features
Advanced Features
Generated Classes
Main frame styles:
Thick frame
Minimize box
Maximize box
Minimized
Maximized
System menu
About box
Initial status bar
Split window
Child frame styles:
Child minimize box
Child maximize box
Child maximized
Command bar (menu/toolbar/ribbon)
Use a classic menu
Classic menu options
Use a classic docking toolbar
Menu bar and toolbar options:
User-defined toolbars and images
Personalized menu behavior
Dialog title
Sketcher
Previous
Next
Finish
Cancel



In the file ***projectname*.cpp** find the function **InitInstance**. You should see the following lines toward the end of the function:

```
        // The one and only window has been initialized, so show and update it
        m_pMainWnd->SetWindowText("Sketcher");
```

```
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();
```

Exercise 5: Sketcher (Part 1) - a drawing program

Drawing Basics

Before you begin writing a graphics program it will help to have a little background on the organization and structure of the underlying graphics system on a PC. When you are drawing on the screen or printing a drawing your code is never interacting directly with the hardware (monitor or printer). There are instead a couple of operating system layers of code between your graphics program code and the hardware device.

Graphics Device Interface (GDI) - This is a device independent layer that is part of the operating system which allows you to do drawing in a window without having to worry about which hardware (screen or printer) a drawing is to be displayed on or how the hardware displays the graphics.

Device Context (DC) - This is a data structure defined by Windows that allows you to send drawing commands to a GDI. It is the actual drawing interface you will use to create your graphics.

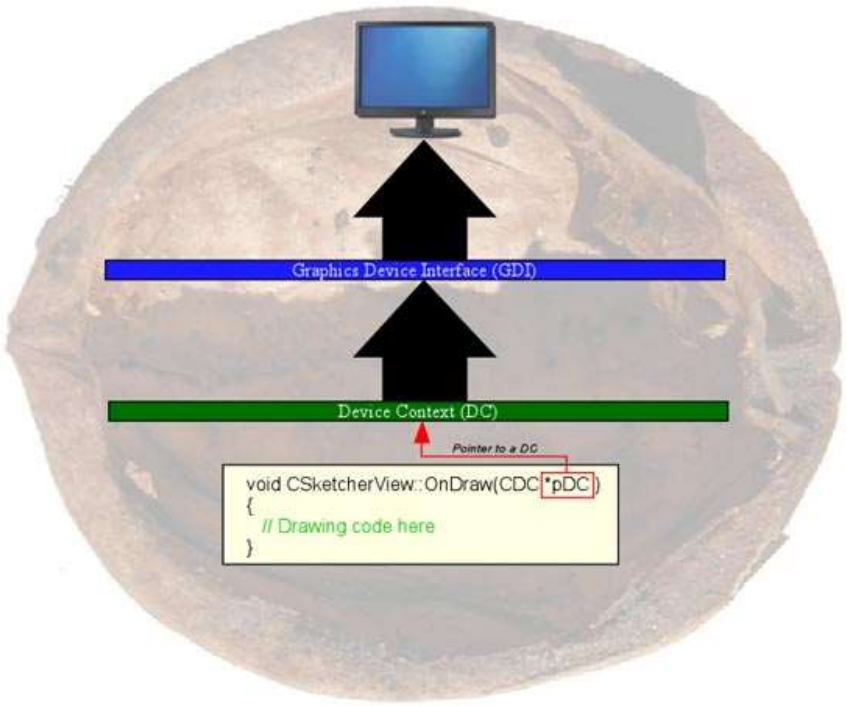
FYI: Mapping Modes - There are a number of, so called, **mapping modes** for a DC all of which define the directions of the X and Y axes and the units used (pixels, inches, etc.) You will only need to be concerned with the **MM_TEXT** mode which uses pixels as the units and sets position (0,0) in the upper left corner of the window client area with X increasing to the right and Y increasing down. This, fortunately is the default mapping mode so you do not have to set anything in your program to handle this.

OnDraw() - If you look in the SketcherView.cpp class file you will find a function called **OnDraw()**. This is where you handle all of the drawing in an application. This function gets called every time there is a WM_PAINT message sent to your application. This will happen when the OS determines that you need to repaint the screen, which may happen as a result of the window being resized, another window being moved over the application window, or any number of other events. What this implies is you can't just draw in a window and forget it, expecting everything to remain the same. Events change things.

Take a look at the OnDraw() function now:

```
void CSketcherView::OnDraw(CDC *pDC);
```

(Note: when the View class is created this is defined as **OnDraw(CDC* /*pDC*/)** so you can give the Device Context pointer any name you want. For now just uncomment and use the default **pDC** name.)



Drawing in a Nutshell

Note: In each of the code samples given below the pointer variable pDC is the pointer to the Device Context which is passed into the OnDraw() function.

Set Window Title

1. In **Sketcher.cpp** locate the function **InitInstance().**
2. At the bottom of the function just after the lines...

```
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();
```

Add the line...

```
m_pMainWnd->SetWindowTextA("Sketcher");
```

Drawing Reference

Below is a brief outline of all of the drawing functions you will need for this exercise. To get started try experimenting with a few of these just to see what happens. Put all of the drawing code in the OnDraw() function found in SketcherView.cpp

Some useful structures and classes

```
struct POINT          class CPoint              class CRect
{                     {                         {
   LONG x;               // Other stuff             // Other stuff
   LONG y;               LONG x;                   LONG left;
};                       LONG y;                   LONG top;
                       };                          LONG right;
                                                   LONG bottom;
class CPen            class CBrush              };
```

Do not define these in your code. They have already been defined in the Device Context.Setting the Current Position

The DC maintains a "current position" point which is the starting point for drawing lines. This can be used and moved.

Function Prototypes

```
CPoint MoveTo(int x, int y);
CPoint MoveTo(POINT pt);
CPoint MoveTo(CPoint pt);
```

Do not prototype these functions in your code. They have already been defined in the Device Context.

Any of these functions moves the current drawing point to the specified (x,y) position. Each returns the position before the move.

Sample code

```
pDC->MoveTo(50, 50);

POINT pt1;
pt1.x = 50;
pt1.y = 50;
pDC->MoveTo(pt1);

CPoint pt2;
pt2.x = 50;
pt2.y = 50;
pDC->MoveTo(pt2);
```

Drawing Lines

Any of these functions draws a line from the last current position to the given position and moves the current position to that point. The function returns TRUE if the line was drawn or FALSE if not.

Function Prototypes

```
BOOL LineTo(int x, int y);
BOOL LineTo (POINT pt);
BOOL LineTo (CPoint pt);
```

Do not prototype these functions in your code. They have already been defined in the Device Context.

Sample code

```
pDC->LineTo(100, 100);

POINT pt1;
pt1.x = 100;
pt1.y = 100;
pDC->LineTo(pt1);

CPoint pt2;
pt2.x = 100;
pt2.y = 100;
pDC->LineTo(pt2);
```

Drawing Rectangles

Any of these functions draws a rectangle with the upper left corner at (x1, y1) and lower right corner at (x2, y2) or it draws a rectangle based on the (left, top, right, bottom) coordinates of the CRect object. The data type LPCRECT is CRect* (pointer to a CRect). Calling either of these functions does not move the current position. The functions returns TRUE if the rectangle was drawn or FALSE if not.

Function Prototypes

```
BOOL Rectangle(int x1, int y1, int x2, int y2);
BOOL Rectangle(LPCRECT rect);
```

Do not prototype these functions in your code. They have already been defined in the Device Context.

Sample code

```
pDC->Rectangle(100, 100, 200, 200);

LPCRECT rect = new CRect(100, 100, 200, 200);
pDC->Rectangle(rect);
```

Drawing Ellipses

Any of these functions draws an ellipse in a bounding rectangle with the upper left corner at (x1, y1) and the lower right corner at (x2, y2) or defined by the CRect object. If the width and height of the bounding rectangle are the same it draws a circle. Calling either of these functions does not move the current position. The functions returns TRUE if the ellipse was drawn or FALSE if not.

Function Prototypes

```
BOOL Ellipse(int x1, int y1, int x2, int y2);
BOOL Ellipse(LPCRECT rect);
```

Do not prototype these functions in your code. They have already been defined in the Device Context.

Sample code

```
pDC->Ellipse(100, 100, 200, 200); // Draw a circle

LPCRECT rect = new CRect(100, 100, 300, 200);
pDC->Ellipse(rect);  // Draws an ellipse
```

Drawing Arcs

Any of these functions draws an arc in a bounding rectangle with the upper left corner at (x1, y1) and the lower right corner at (x2, y2) starting from (x3, y3) going to (x4, y4) or using the corresponding CRect and POINT/CPoint objects. If the start and end points are the same it draws an arc of 360 degrees that appears as a circle. Calling either of these functions does not move the current position. The functions returns TRUE if the arc was drawn or FALSE if not.

Lines from the bounding rectangle center to the StartPt and EndPt defines range of the arc. The start and end point do not have to be on the bounding rectangle edge or the arc edge.

Function Prototypes

```
BOOL Arc(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);
BOOL Arc (LPCRECT rect, POINT StartPt, POINT EndPt);
BOOL Arc (LPCRECT rect, CPoint StartPt, CPoint EndPt);
```

Do not prototype these functions in your code. They have already been defined in the Device Context.

Sample code

```
pDC->Arc(50, 50, 150, 150, 125, 50, 100, 150);

LPCRECT rect = new CRect(50, 50, 150, 150);
POINT pt1;
pt1.x = 125;
pt1.y = 50;
POINT pt2;
pt2.x = 100;
pt2.y = 150;
pDC->Arc(rect, pt1, pt2);

LPCRECT rect2 = new CRect(250, 250, 350, 350);
CPoint pt3;
```

```
pt3.x = 225;
pt3.y = 150;
CPoint pt4;
pt4.x = 200;
pt4.y = 250;
pDC->Arc(rect, pt3, pt4);
```

Note: Arc

draws an arc but does not fill it with the brush color. If you want a filled arc use the function

Pie

. It takes the exact same function arguments as

Arc

.

Drawing and Filling in Color

You can define a pen with which to draw by using the

CPen

class. You can define a pattern for filling enclosed elements (rectangles, ellipses, arcs, etc.) by using the

CBrush

class.

CPen

You can create a CPen object and initialize it at the same time or you can just create a CPen object and then call the

CreatePen

method to initialize it.

Function Prototypes

```
CPen(int penStyle, int width, COLORREF COLOR);  // CPen default constructor
BOOL CreatePen(int penStyle, int width, COLORREF color); // Initializes a CPen object
```

Do not prototype these functions in your code. They have already been defined in the Device Context.

Sample code

```
CPen aPen;              // Create a pen
aPen.CreatePen(PS_SOLID, 2, RGB(255,0,0));      // Init the pen to draw lines solid, 2 pixels wide, and red

CPen aPen(PS_SOLID, 2, RGB(255,0,0)); // Create and init a pen
```

Pen Styles

- PS_SOLID
- PS_DASH
- PS_DOT
- PS_DASHDOT
- PS_DASHDOTDOT
- PS_NULL
- PS_INSIDEFRAME (like solid but points specified are on the edge of the pen width instead of in the center)

CBrush

You can create a CBrush object and initialize it at the same time or you can just create a CBrush object and then call the

CreateBrush

method to initialize it.

In the sample code below a bitmap brush is created using the DukeBrush.bmp image. You can download a copy of this bitmap image by saving the image shown here.

Function Prototypes

```
CBrush(int brushStyle, COLORREF color);
BOOL CreateBrush(int brushStyle, COLORREF color);
```

Do not prototype these functions in your code. They have already been defined in the Device Context.

Sample code

```
// Create a solid red brush
CBrush aBrush(RGB(255, 0, 0));

// Create a solid red brush
CBrush aBrush;
aBrush.CreateSolidBrush(RGB(255, 0, 0));

// Create a horizontal hatch red brush
CBrush aBrush(HS_HORIZONTAL, RGB(255, 0, 0));

// Create a horizontal hatch red brush
CBrush aBrush;
aBrush.CreateHatchBrush(HS_HORIZONTAL, RGB(255, 0, 0)); // Init the brush

// Create a brush using a bitmap
CBrush aBrush;
CBrush bmpBrush;
HBITMAP hBmp = (HBITMAP)LoadImage(NULL,"DukeBrush.bmp",IMAGE_BITMAP,0,0,
               LR_LOADFROMFILE|LR_DEFAULTSIZE);
if(hBmp != NULL)
{
    CBitmap Bmp;
    Bmp.Attach(hBmp);
    bmpBrush.CreatePatternBrush(&Bmp);
}
else
{
    aBrush.CreateSolidBrush(RGB(0, 0, 255));
}
```

Bitmap brushes will tile the bitmap inside the object being filled.

Brush Styles

- HS_HORIZONTAL
- HS_VERTICAL
- HS_BDIAGONAL
- HS_FDIAGONAL
- HS_CROSS
- HS_DIAGCROSS
- HS_NULL
- PS_INSIDEFRAME (like solid but points specified are on the edge of the pen width instead of in the center)

Selecting which CPen and CBrush to use

```
    pDC->SelectObject(&aPen);          // Assumes aPen is a CPen object not a pointer
```

```
pDC->SelectObject(&aBrush);        // Assumes aBrush is a CBrush object not a pointer
```

Saving and restoring pens and brushes

```
CPen *oldPen = pDC->SelectObject(aPen);
// Use the pen
pDC->SelectObject(oldPen);     // Restore original pen

CBrush *oldBrush = pDC->SelectObject(aBrush);
// Use the brush
pDC->SelectObject(oldBrush);     // Restore original brush
```

The sample code below is from the OnDraw() function in a Sketcher demonstration program. The image below the code shows the results of these drawing commands. You can copy the code directly into your OnDraw() function in

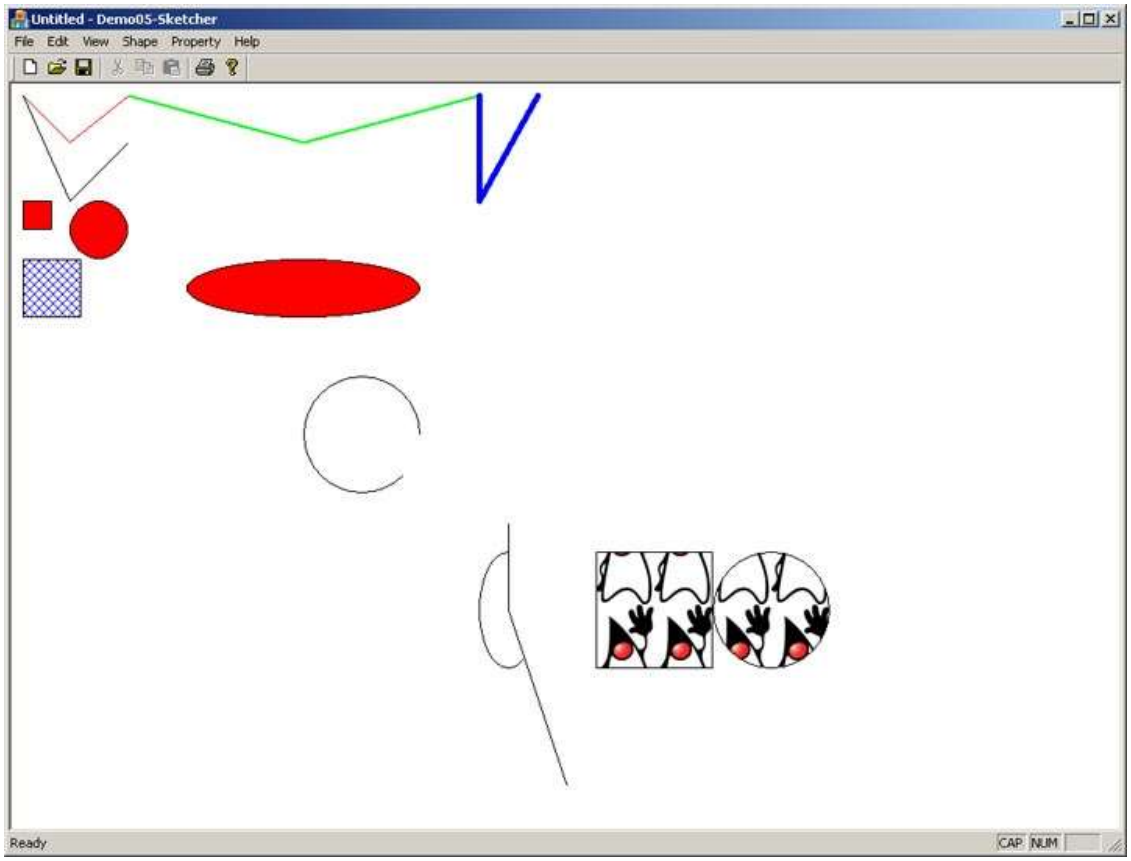
***projectName*View.cpp**

. Note the comments and try some variations on these drawing commands.

```
void CDemo05SketcherView::OnDraw(CDC* pDC)
{
        CDemo05SketcherDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        if (!pDoc)
                return;

        // TODO: add draw code for native data here

        // Demonstration drawing lines with default pen
        pDC->MoveTo(10, 10);
        POINT pt1;
        pt1.x = 50;
        pt1.y = 100;
        pDC->LineTo(pt1);
        CPoint pt2;
        pt2.x = 100;
        pt2.y = 50;
        pDC->LineTo(pt2);

        // Define some pens and draw lines.  Note: we save and
        //  restore the default pen.  You should also delete any
        //  pen you create, BUT NOT WHILE IT IS SELECTED IN THE DC.
        // Pen Styles: PS_SOLID, PS_DASH, PS_DOT, PS_DASHDOT,
        // PS_DASHDOTDOT, PS_NULL, PS_INSIDEFRAME
        CPen redPen(PS_SOLID, 1, RGB(255, 0, 0)); // Create solid red pen in one setp
        CPen* pOldPen = pDC->SelectObject(&redPen); // Set the pen to draw with
        pDC->MoveTo(10, 10);
        pDC->LineTo(50, 50);
        pDC->LineTo(100, 10);
        // Create dash green pen, 2 pixels wide, in 2 steps
        CPen greenPen;
        greenPen.CreatePen(PS_DASH, 2, RGB(0, 255, 0));
        pDC->SelectObject(&greenPen);
        pDC->LineTo(250, 50);
        pDC->LineTo(400, 10);
        // Create a dotted, 5 pixels wide blue pen
        CPen bluePen(PS_DOT, 5, RGB(0, 0, 255));
        pDC->SelectObject(&bluePen);
```

```
        pDC->LineTo(400, 100);
        pDC->LineTo(450, 10);
        // Restore default pen
        pDC->SelectObject(pOldPen);
        // Delete the pens used. DO NOT DO THIS WHILE A PEN IS STILL
        //  SELECTED IN THE DC
        redPen.DeleteObject();
        greenPen.DeleteObject();
        bluePen.DeleteObject();

        // Create a brush for filling shapes
        // Styles: HS_HORIZONTAL, HS_VERTICAL, HS_BDIAGONAL, HS_FDIAGONAL,
        // HS_CROSS, HS_DIAGCROSS
        // To create an empty brush use...
        // CBrush nBrush; // = pDC->CreateStockObject(NULL_BRUSH);
        // nBrush.CreateStockObject(NULL_BRUSH);
        // pDC->SelectObject(nBrush);

        // Create a solid red brush and fill 2 rectangles with it.
        // The outline is drawn with the currently set pen.
        // Note: We save the default brush for later resetting
        CBrush redBrush(RGB(255, 0, 0));
        CBrush *pOldBrush = pDC->SelectObject(&redBrush);
        LPCRECT rect = new CRect(10, 100, 35, 125);
        // Note: LPCRECT is a pointer and the rectangle args define
        // left, top, right, bottom coordinates.
        pDC->Rectangle(rect);
        // Create a blue brush with Diagonal Cross Hatching
        CBrush blueBrush(HS_DIAGCROSS, RGB(0, 0, 255));
        pDC->SelectObject(&blueBrush);
        pDC->Rectangle(10, 150, 60, 200);
        delete rect; // Get rid of the rectangle object

        // Reset to redBrush
        pDC->SelectObject(&redBrush);
        blueBrush.DeleteObject();  // Get rid of the blue brush

        // Draw a couple of ellipses (ovals). Note if width and
        //  height are equal the ellipse will be a circle
        pDC->Ellipse(50, 100, 100, 150);
        LPCRECT rect2 = new CRect(150, 150, 350, 200);
        pDC->Ellipse(rect2);
        delete rect2;

        // Draw a couple of arcs, one filled with the red brush
        //  another filled with the default brush and lines drawn
        //  from the center to the defining points
        //BOOL Arc (LPCRECT rect, POINT StartPt, POINT EndPt);
        //BOOL Arc (LPCRECT rect, CPoint StartPt, CPoint EndPt);
        //BOOL Arc (int left, int top, int right, int bottom,
        //          int startX, int startY, int endX, int endY);
        LPCRECT rect3 = new CRect(250, 250, 350, 350);
        CPoint pt3, pt4;
        pt3.x = 350;
        pt3.y = 300;
        pt4.x = 350;
        pt4.y = 350;
        pDC->Arc(rect3, pt3, pt4);
        pDC->Arc(400, 400, 450, 500, 425, 375, 475, 600);
        // Note these lines show where the actual start and end points
        //  are by drawing lines from the center of the arc to the points.
        // Drawing is done in a counter-clockwise direction
        pDC->MoveTo(425, 450);
        pDC->LineTo(425, 375);
        pDC->MoveTo(425, 450);
        pDC->LineTo(475, 600);
        delete rect3;
```

```
        // Create a bitmap brush
        CBrush bmpBrush;

        // Put a .bmp file in the same directory as the source for this
        //   file and change the name in the line below.
        HBITMAP hBmp = (HBITMAP)LoadImage(NULL,"DukeBrush.bmp",IMAGE_BITMAP,0,0,
                LR_LOADFROMFILE|LR_DEFAULTSIZE);
        if(hBmp != NULL)
        {
                CBitmap Bmp;
                Bmp.Attach(hBmp);
                bmpBrush.CreatePatternBrush(&Bmp);
                pDC->SelectObject(bmpBrush);
        }
        // If creation of bitmap brush failed we will just use
        //  the previously set brush.

        // Draw a rectangle and an oval with the bitmap brush
        pDC->Rectangle(500, 400, 600, 500);
        pDC->Ellipse(600, 400, 700, 500);

        // Clean up
        pDC->SelectObject(pOldBrush); // Restore default brush, already did pen
        redBrush.DeleteObject();
        bmpBrush.DeleteObject();
}
```



When you finish experimenting with the drawing functions be sure you remove or comment out all the code you added to the OnDraw() function. We don't want that interfering with the drawing program later.