More detailed notes at:
http://www.cs.binghamton.edu/~reckert/360/class15.htm

MFC Windows Programming:
App/Window vs. Document/View
Approach

- An App/Window approach program creates application and window objects
- Mirrors Win32 API program organization
- Main difference--MFC automates and masks details … and does many other necessary tasks
- But data & rendering of data are intertwined
- Frequently, data members exist in window class
  - Example in MSG2005.CPP: Output string defined in window-based class
    - But output string is data
    - Really has nothing to do with window it’s being displayed in
Conceptually data is different from rendering of data
In an App/Window approach program they are mixed together in same window class
Frequently we need to have different views of same data
  – (e.g., displaying data in a window or on a printer)
So it’s a good idea to separate data and data presentation

Doc/View Achieves Separation of Data and Data Presentation
- Encapsulates data in a CDocument class object
- Encapsulates data display and user interaction with it in a CView class object
- Classes derived from CDocument
  – Should handle anything affecting an application’s data
- Classes derived from CView
  – Should handle display of data and user interactions with that display
Other Classes are Still Needed

Still need to create *CFrameWnd* and *CWinApp* classes
But their roles are reduced

Documents

**Document**
- Contain any forms of data associated with the application (pure data)
- Not limited to text
- Could be anything
  - game data, graphical data, etc.
Views

- A rendering of a document; a physical representation of the data
- Provides mechanism for displaying data stored in a document
- Defines how data is to be displayed in a window
- Defines how the user can interact with it

Frame Window

- Window in which a view of a document is displayed
- A document can have multiple views associated with it
  - different ways of looking at the same data
- But a view has only one document associated with it
MFC Template Class Object

- Handles coordination between documents, views, and frame windows

- In general:
  - Application object creates a template...
  - which coordinates display of document's data...
  - in a view...
  - inside a frame window

- i.e., our CWinApp object creates a Document Template which creates a CDocument object and a CFrameWnd object
  - The CFrameWnd object creates a CView object
  - Which displays the document data
Dynamic Creation

- In Doc/View approach, objects are dynamic
- Doc/View program is run
  - Its frame window, document, and view are created dynamically
  - Often Doc/View objects are synthesized from file data
    - They need to be created at load time (run time)
  - To allow for dynamic creation, use dynamic creation macros
    - in classes derived from CFrameWnd, CDocument, and CView

Relationship between Application, Document Template, Document, Frame Window, & View in a Document/View Approach MFC Program.
Document/View Programs

Almost always have at least four classes derived from:
- `CFrameWnd`
- `CDocument`
- `CView`
- `CWinApp`

Usually put into separate declaration (.h) and implementation (.cpp) files
Because of template and dynamic creation, there’s lots of initialization
Could be done by hand, but nobody does it that way

Microsoft Developer Studio
AppWizard and ClassWizard Tools
AppWizard

- Tool that generates a Doc/View MFC program framework automatically
- Can be built on and customized by programmer
- Fast, efficient way of producing Windows Apps
- Performs required initialization automatically
- Creates functional `CFrameWnd, CView, CDocument, CWinApp` classes
- After AppWizard does it’s thing:
  - Application can be built and run
  - Full-fledged window with all common menu items, tools, etc.

ClassWizards

- Facilitate message handling in a framework-based MFC application
- Tools that connect resources and user-generated events to program response code
- Write C++ skeleton routines to handle messages
- Insert code into appropriate places in program
  - Code then can then be customized by hand
- Can be used to create new classes or derive classes from MFC base classes
  - Add new member variables/functions to classes
- In .NET many “class wizards” are available through Properties window
SKETCH Application

- Example of Using AppWizard and ClassWizard
- User can use mouse as a drawing pencil
  - Left mouse button down:
    - lines in window follow mouse motion
  - Left mouse button up:
    - sketching stops
- User clicks "Clear" menu item
  - window client area is erased

- Sketch data (points) won’t be saved
  - So leave document (**CSketchDoc**) class created by AppWizard alone
- Base functionality of application (**CSketchApp**) and frame window (**CMainFrame**) classes are adequate
  - Leave them alone
- Use ClassWizard to add sketching to **CView** class
Sketching Requirements

➔ If left mouse button is down:
  – Each time mouse moves:
    • Get a DC
    • Create a pen of drawing color
    • Select pen into DC
    • Move to old point
    • Draw a line to the new point
    • Make current point the old point
    • Select pen out of DC

 Variables

➔ BOOLEAN m_butdn
➔ CPoint m_pt, m_ptold
➔ COLORREF m_color
➔ CDC* pDC
Steps in Preparing SKETCH

1. “File / New / Project”
   - Project Type: “Visual C++ Projects”
   - Template: “MFC Application”
   - Enter name: Sketch

2. In “Welcome to MFC Application Wizard”
   - Application type: “Single Document” Application
   - Take defaults for all other screens

3. Build Application --> Full-fledged SDI App with empty window and no functionality

4. Add member variables to CSketchView
   - Can do manually in .h file
   - Easier to:
     • Select Class View pane
     • Click on SketchView class
       - Note member functions & variables
     • Right click on CSketchView class
       - Choose “Add” / “Variable”
         - Launches “Add Member Variable Wizard”
       - Variable Type: enter CPoint
       - Name: m_pt
       - Access: Public (default)
         - Note after “Finish” that it's been added to the .h file
     • Repeat for other variables (or add directly in .h file):
       - CPoint m_ptold
       - bool m_butdn
       - COLORREF m_color
       - CDC* pDC
5. Add message handler functions:
   – Select CSketchView in Class View
   – Select “Messages” icon in Properties window
     • Results in a list of WM_ messages
   – Scroll to WM_LBUTTONDOWN & select it
   – Add the handler by clicking on down arrow and
     “<Add> OnLButtonDown”
     • Note that the function is added in the edit window and the
cursor is positioned over it:
     – After “TODO…” enter following code:
       m_butdn = TRUE;
       m_ptold = point;
Repeat process for WM_LBUTTONDOWN handler:

- Scroll to WM_LBUTTONDOWN
- Click: “<Add> OnLButtonDown”,
- Edit Code by adding:
  ```cpp
  m_butdn = FALSE;
  ```
Repeat for WM_MOUSEMOVE
- Scroll to WM_MOUSEMOVE
- Click: "<Add> OnMouseMove"
- Edit by adding code:
  if (m_butdn)
  {
    pDC = GetDC();
    m_pt = point;
    CPen newPen (PS_SOLID, 1, m_color);
    CPen* pPenOld = pDC->SelectObject (&newPen);
    pDC->MoveTo (m_ptold);
    pDC->LineTo (m_pt);
    m_ptold = m_pt;
    pDC->SelectObject (pPenOld);
  }

6. Initialize variables in CSketchView constructor
- Double click on CSketchView constructor
  • CSketchView(void) in Class View
- After "TODO…", Add code:
  m_butdn = FALSE;
  m_pt = m_ptold = CPoint(0,0);
  m_color = RGB(0,0,0);
7. Changing Window’s Properties
   – Use window’s SetWindowXxxxx() functions
     • In CWinApp-derived class before window is shown and updated
   – Example: Changing the default window title
     
     ```cpp
     m_pMainWnd->SetWindowText(TEXT("Sketching Application"));
     ```
   – There are many other CWnd SetWindowXxxxx() functions that can be used to change other properties of the window

8. Build and run the application

Menus and Command Messages

- User clicks on menu item
- WM_COMMAND message is sent
- ID XXX identifies which menu item (its ID)
- No predefined handlers
- So message mapping macro is different
- ON_COMMAND(ID XXX, OnXxx)
  - OnXxx() is the handler function
  - Must be declared in .h file and defined in .cpp file
Adding Color and Clear Menu Items to SKETCH App

Resource View (sketch.rc folder)
- Double click Menu folder
- Double click IDR_MAINFRAME menu
- Add: “Drawing Color” popup menu item with items:
  - “Red”, ID_DRAWING_COLOR_RED (default)
  - “Blue”, ID_DRAWINGCOLOR_BLUE
  - “Green”, ID_DRAWINGCOLOR_GREEN
  - “Black”, ID_DRAWINGCOLOR_BLACK
- Add another main menu item:
  - “Clear Screen”, ID_CLEARSCREEN
  - Set Popup property to False

Add Menu Item Command Handler Function

- One way: Use “Event Handler Wizard”
- In “Resource View” bring up menu editor
- Right click on “Red” menu item
- Select “Add Event Handler” “Event Handler Wizard” dialog box
  - Class list: CSketchView
  - Message type: COMMAND
  - Function handler name: accept default
    - OnDrawingcolorRed
  - Click on “Add and edit”
  - After “TODO…” in editor enter following code:
    
m_color = RGB(255,0,0);
In Class View Select CSketchView

- In Properties window select Events (lightning bolt icon)
- Scroll down to: ID_DRAWINGCOLOR_RED
- Select “COMMAND”
- Click “<Add> OnDrawingcolorRed” handler
- Edit code by adding:
  
  ```
  m_color = RGB(255,0,0);
  ```

Another Method of Adding a Menu Item Command Handler

- In Class View Select CSketchView
- In Properties window select Events (lightning bolt icon)
- Scroll down to: ID_DRAWINGCOLOR_RED
- Select “COMMAND”
- Click “<Add> OnDrawingcolorRed” handler
- Edit code by adding:
  
  ```
  m_color = RGB(255,0,0);
  ```
Repeat for ID_DRAWINGCOLOR_BLUE
Code: m_color = RGB(0,0,255);

Repeat for ID_DRAWINGCOLOR_GREEN
Code: m_color = RGB(0,255,0);

Repeat for ID_DRAWINGCOLOR_BLACK
Code: m_color = RGB(0,0,0);

Repeat for ID_CLEAR
Code: Invalidate();
Destroying the Window

- Just need to call *DestroyWindow()*
  - Do this in the CMainFrame class – usually in response to a “Quit” menu item