MFC Windows Programming: Document/View Approach

MFC Windows Programming: Document/View Approach

- App/Window approach creates application and window objects
- Mirrors Win32 API program organization
- Main difference--MFC automates & masks details
- But data & rendering of data are intertwined
- Frequently, data members exist in window class
  - Example in MSG1.CPP: Output string & position both defined in window-based class
    - Output string is data
    - Position is user defined

- Conceptually data is different from rendering of data
- In an App/Window they are mixed together in same window class
- Frequently need to have different views of same data
  - (e.g., displaying data in a window or on a printer)
- So it would be good to separate data and data presentation

Doc/View Achieves Separation

- Encapsulates data in a ***CDocument*** class object
- Encapsulates data display mechanism data 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 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.

Document Interfaces

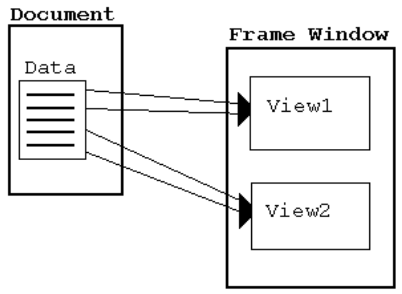
- **Single Document interface** (**SDI**) application
  - Program that deals with one document at a time
  - All our programs to date have been **SDI** apps
- **Multiple Document Interface** (**MDI**) application
  - Program organized to handle multiple documents simultaneously
  - Multiple open documents can be of same or different types
  - Example of an **MDI** application: Microsoft Word

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

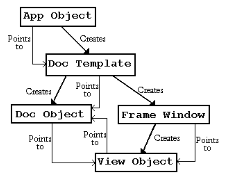


Documents, Views, & Frames

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

Template/Document/View/Window



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

Serialization

- Provides for storage/retrieval of document data
- Usually to/from a disk file
- ***CDocument*** class has serialization built into it
  - So in DOCUMENT/VIEW apps, saving/storing data is straightforward

Dynamic Creation

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

Dynamic Creation Macros

- ***DECLARE_DYNCREATE(class_name)***
  - in declaration (.h file)
- ***IMPLEMENT_DYNCREATE(class_name, parent_class_name)***
  - (in .cpp file)
- ***After IMPLEMENT_DYNCREATE()*** macro is invoked:
  - Class is enabled for dynamic creation
  - Now a template can be created

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.

ClassWizard

- Message handling in a framework-based MFC application facilitated by using ClassWizard
- A tool that connects resources & user-generated events to program response code
- Writes C++ skeleton routines to handle messages
- Inserts 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

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 / MFC AppWizard (exe)
  - Enter name: Sketch
  - Step 1: Choose “Single Document” (SDI App)
  - Take defaults for Steps 2-6
- 2. Build App --> Full-fledged SDI App with empty window and no functionality
- 3. Add member variables to CSketchView
  - Can do manually in .h file

- 3. Easier to:
  - Select ClassView tab and expand (+)
    - Note member functions & variables
  - Right click on CSketchView
    - Choose “Add member variable”
    - Type: CPoint
    - Name: m_pt
    - Access: Public (default)
  - Repeat for:
    - CPoint m_ptold
    - BOOL m_butdn
    - COLORREF m_color
    - CDC* pDC

- 4. Use ClassWizard (Icon or Ctrl-w) to set up message map and handler function
  - Message Maps tab
  - Class name: CSketchView
  - Object ID: CSketchView highlighted
  - Messages:
    - Scroll to WM_LBUTTONDOWN
    - Click: Add Function, Edit Code:
      - After “TODO…” enter following code:

```
m_butdn = TRUE;
m_ptold = point;
```

- Repeat process for WM_LBUTTONUP handler
  - Scroll to WM_LBUTTONUP
  - Click: Add Function, Edit Code:
  - Enter:

```
m_butdn = FALSE;
```

- Repeat for WM_MOUSEMOVE
  - Scroll to WM_MOUSEMOVE
  - Click: Add Function, Edit 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);
}
```

- 5. Initialize variables in CSketchView constructor
  - Double click on CSketchView constructor (Classview)
  - After “TODO…”, Add code:

```
m_butdn = FALSE;
m_pt = m_ptold = CPoint(0,0);
m_color = RGB(0,0,0);
```

- 6. Build Project and Run

Menus and Command Messages

- User clicks on menu item
- WM_COMMAND message sent
- IDM_XXX identifies which menu item
- No predefined handlers
- So message mapping macro is different
- ON_COMMAND(IDM_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

- 1. Resource View (Sketch resources)
  - Double click menu
  - Double click IDR_MAINFRAME menu
  - Add: “Drawing Color” popup menu item with items:
    - IDM_RED: “Red”
    - IDM_BLUE: “Blue”
    - IDM_GREEN: “Green”
    - IDM_BLACK: “Black”
    - IDM_CLEAR: “Clear Screen”

- 2. Add menu item command handler functions (message map)
  - ClassWizard (Ctrl-w or icon)
    - Class name: CSketchView
    - ObjectID: Select IDM_BLACK
    - Messages: Select COMMAND
    - AddFunction / OK / Edit Code
    - After “TODO…” enter ff. Code:
      m_color = RGB(0,0,0);

  Repeat for IDM_BLUE, Code: m_color = RGB(0,0,255);
  Repeat for IDM_GREEN, Code: m_color = RGB(0,255,0);
  Repeat for IDM_RED, Code: m_color = RGB(255,0,0);
  Repeat for IDM_CLEAR, Code: Invalidate();