Win32 API Programming

- See also the old notes at:

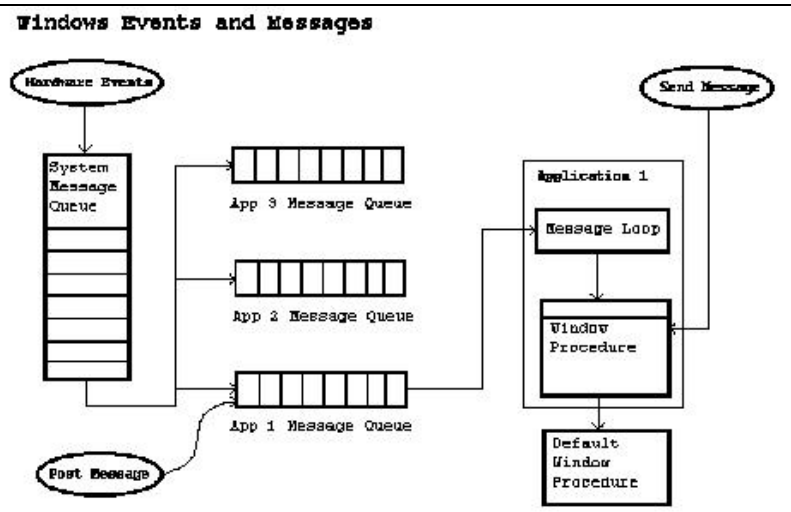
  http://www.cs.binghamton.edu/~reckert/360/class2a.htm
  http://www.cs.binghamton.edu/~reckert/360/class3a.htm

(C) Richard R. Eckert

Win32 API Programming

- Event-driven, graphics oriented
- Example: User clicks mouse over a program's window area (an event) --
  - Windows decodes HW signals from mouse
  - figures out which window user has selected
  - sends a message to that window's program:
    - "User has clicked over (X,Y)"
    - "Do something and return control to me"
  - Program reads message data, does what's needed, returns control to Windows

(C) Richard R. Eckert



Overview of Win32 API Program Structure--2 main tasks:

- Initial activities
- Process messages from Windows
  - the message loop

(C) Richard R. Eckert

Pseudocode

- Initialize variables, memory space
- Create and show program's window
- Loop
  - Fetch any message sent from Windows to this program
  - If message is WM_QUIT
    - terminate program, return control to Windows
  - If message is something else
    - take actions based on message and parameters
    - return control to Windows
- End Loop

(C) Richard R. Eckert

Essential Parts of a Win32 API Pgm

- I. The source program (.c/.cpp file):
  - A. WinMain() function
    - 0. declarations, initialization, etc.
    - 1. register window "class"
    - 2. create a window based on a registered "class"
    - 3. show window, make it update its client area
    - 4. the message loop (get messages from Windows, dispatch back to Windows for forwarding to correct callback message-processing function, the WndProc)
  - B. WndProc(): the message-processing function
    - we write this function

(C) Richard R. Eckert

- II. The resource script (.rc file):
  - contains resource (Windows static) data
  - separate from code and dynamic data
  - compiled by a separate "Resource Compiler"
  - Resources determine the app's "look and feel"
  - Examples of Windows Resources:
    - Keyboard Accelerators, Bitmaps, Cursors, Dialog Box specs, Fonts, Icons, Menus, String Tables
  - Separation of resources and program code==>
    - separates tasks of programmer & designer
    - can change user interface without touching code

(C) Richard R. Eckert

The WinMain( ) Function

- int PASCAL WinMain (HINSTANCE hInstance,
  HINSTANCE hPrevInstance,
  LPSTR lpszCmdLine, int nCmdShow);
- WinMain() starts first
- integer exit code returned to Windows OS
- PASCAL: l-to-r parameter passing on stack
- 4 parameters passed in from Windows
  - hInstance: a "handle", identifies current pgm instance
  - lpszCmdLine: string containing command line args
  - nCmdShow: how window is to appear when shown

(C) Richard R. Eckert

Hungarian Notation

- help clarify variable types
- precede name with key letters representing type
- named after Hungarian Microsoft programmer, Charles Simonyi

(C) Richard R. Eckert

```
prefix  data type
 ---------------------------------
 by   BYTE (unsigned char)
 b    BOOL (int, TRUE=1 FALSE=0)
 c    char
 dw   DWORD (4-byte unsigned long)
 fn   function
 h    handle
 l    long (4 bytes)
 n    short (int) near pointer
 p    pointer
 sz   null-terminated char string
 w    word (two bytes)
 lpsz long ptr to null-terminated str
```

(C) Richard R. Eckert

RegisterClass(&wndclass);

```
typedef struct  tagWNDCLASS {
  UINT    style;
  LRESULT CALLBACK lpfnWndProc;
  int     cbClsExtra;
  int     cbWndExtra;
  HINSTANCE  hInstance;
  HICON   hIcon;
  HCURSOR hCursor;
  HBRUSH  hBackground;
  LPCSTR   lpszMenuName;
  LPCSTR   lpszClassName;     }  WNDCLASS;
```

```
WNDCLASS    wndclass;
if (!RegisterClass (&wndclass))  return 0;
```

(C) Richard R. Eckert

CreateWindow( ) arguments:

window class name

window caption

window style (Boolean OR of window style masks)

initial x , y position in pixels

initial width , height

parent window handle (if main window, NULL)

window menu handle (NULL if class menu used)

program instance handle (from Windows)

creation parameters (for extra data, usually NULL)

Returns a handle (hWnd) to the resulting window

(C) Richard R. Eckert

ShowWindow (hWnd,nCmdShow);

- makes window visible on screen
- hWnd: which window to make visible
- nCmdShow: how -- normal, minimized, etc.
  - set by Windows environment when program is started;
  - value is passed in from Windows;

(C) Richard R. Eckert

UpdateWindow (hWnd);

- Causes client area to be updated
- Painted with background brush

(C) Richard R. Eckert

The Message Loop

- User interaction ✍ a message sent to a window
- Lots of other kinds of actions ✍ messages
- A message structure:
  - HWND hwnd; // target window handle
  - UINT message; // message ID value: WM_***
  - WPARAM wParam; // data passed in message
  - LPARAM lParam; // more data in message
  - DWORD time; // time message was sent
  - POINT pt; // mouse cursor position (x,y)

(C) Richard R. Eckert

GetMessage()

- Program must keep checking for messages
- Use message loop with GetMessage()
- *BOOL GetMessage(*

  LPMSG lpMsg, // pointer to message struct

  HWND hWnd, // target window(s)

  // which windows do we want to get msgs from

  UINT wMsg1, // 1st message in range

  UINT wMsg2 // last message in range)

(C) Richard R. Eckert

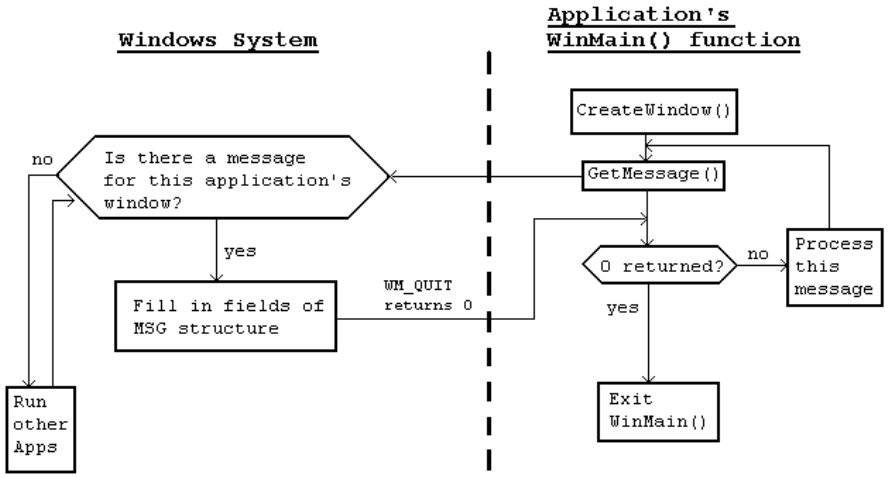
GetMessage( )

- Reads next message from app's message queue
- Fills MSG structure pointed to by first parameter
- Place in a loop:

  while (GetMessage(&msg, NULL, 0, 0))
  { ... }
  return((int)msg.wParam);
- Returns non-0, except for WM_QUIT message
  - – Terminates message loop and returns control to Windows

(C) Richard R. Eckert

The Main Message Loop



(C) Richard R. Eckert

Message Processing

- What goes inside the message loop:

TranslateMessage (&msg)—

– "Cooks" keyboard input

– Converts raw key codes to ANSI codes

– Only important if WM_CHAR message is to be handled

DispatchMessage (&msg)—

– Sends message on to Windows, which

– Forwards it to program's "Window Procedure":

WndProc()—

- 2nd member of WNDCLASS structure
- Programmer must write this function

(C) Richard R. Eckert

The Window Procedure

- "Callback" function (called by Windows)
- Should contain a switch/case statement :
  - Looks at message ID of current message
  - Acts appropriately on "interesting" messages
  - Forwards other messages to default Window procedure--*DefWindowProc()*

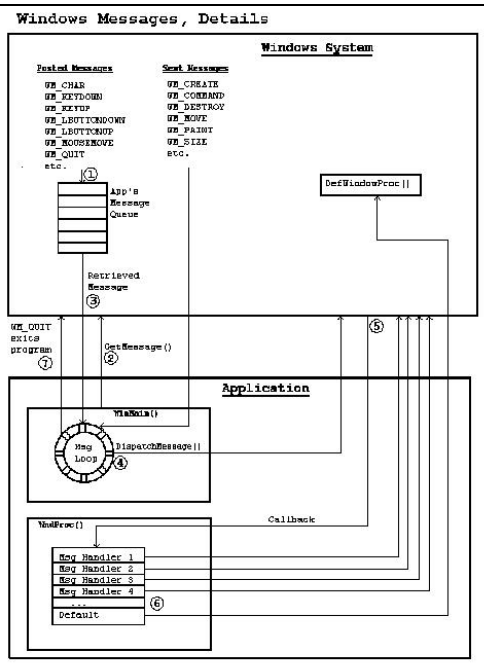
(C) Richard R. Eckert

WndProc()

LRESULT CALLBACK WndProc (
HWND hWnd, UINT wMessage,
WPARAM wParam, LPARAM lParam)

- Parameters—
  - Same as first four fields of MSG structure:
    - window associated with message
    - message ID (what message is)
    - message data (wParam and lParam)
- Return value--
  - Result of message handling
    - 0 means message was handled

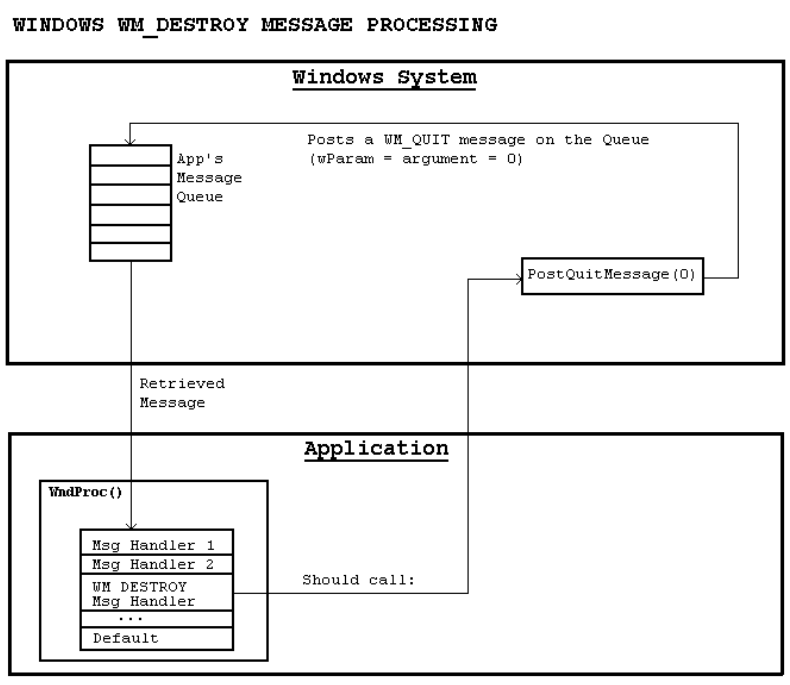
(C) Richard R. Eckert



The WM_DESTROY Message

- Sent by OS when window is destroyed
- WndProc() should respond by calling:
  - *PostQuitMessage();*
  - Windows sends WM_QUIT msg to queue
  - wParam = 0 implies:
    - 0 returned by *GetMessage()* in *WinMain()*
    - normal return to Windows
    - so program exits *WinMain()*'s message loop
    - And returns to Windows

(C) Richard R. Eckert



Example Program

- See First Windows Program: winapp1.cpp
- URL:

http://www.cs.binghamton.edu/~reckert/360/winapp1-cpp.html

(C) Richard R. Eckert

Some other important messages

- WM_COMMAND--User clicked on menu item
  - LOWORD(wParam)=menu item ID
- WM_?BUTTONDOWN--left/right mouse button pressed
  - ? = L, R, or M
  - lParam=x,y coordinates
- WM_MOUSEMOVE--mouse moved
  - lParam=x,y coordinates
- WM_CHAR--User pressed valid ANSI code character or keyboard key combination
  - wParam=ANSI code
- WM_PAINT--window was exposed, should be redrawn
- WM_KEYDOWN--keyboard key pressed
  - wParam=virtual key code

(C) Richard R. Eckert

II. The Resource Script (.rc file)

- Resources--static data
- Example: a menu
- Defined in a script (.rc) file--

```
#include "resource.h"
  MYMENU MENU
  BEGIN
    MENUITEM "&Circle",       ID_CIRCLE
   MENUITEM "&Rectangle",    ID_RECTANGLE
   MENUITEM "Clear &Screen",ID_CLEARSCREEN
   MENUITEM "&Quit",         ID_QUIT
  END
```

(C) Richard R. Eckert

The Resource header (.h file)

```
// resource.h
#define ID_CIRCLE      40006
#define ID_RECTANGLE   40007
#define ID_CLEARSCREEN 40008
#define ID_QUIT        40009
```

- Must be #included in .CPP and .RC files
- Can use Visual Studio's resource editors to prepare .rc and .h visually
  - ID numbers generated automatically

(C) Richard R. Eckert

Key Idea with Menus:

- when menu item is selected
  - Windows sends a WM_COMMAND msg
  - low word of wParam=selected item ID
  - extract with macro LOWORD()
  - then do switch/case on LOWORD(wParam) to perform correct action

(C) Richard R. Eckert

Text and Graphics Output

- Displaying something in a window
- Text and graphics done one pixel at a time
- Any size/shape/position possible
- Design goal: Device Independence

(C) Richard R. Eckert

Device Independent Graphics Interface

- Windows programs don't access hardware devices directly
- Make calls to generic drawing functions within the Windows 'Graphics Device Interface' (GDI) -- a DLL
- The GDI translates these into HW commands

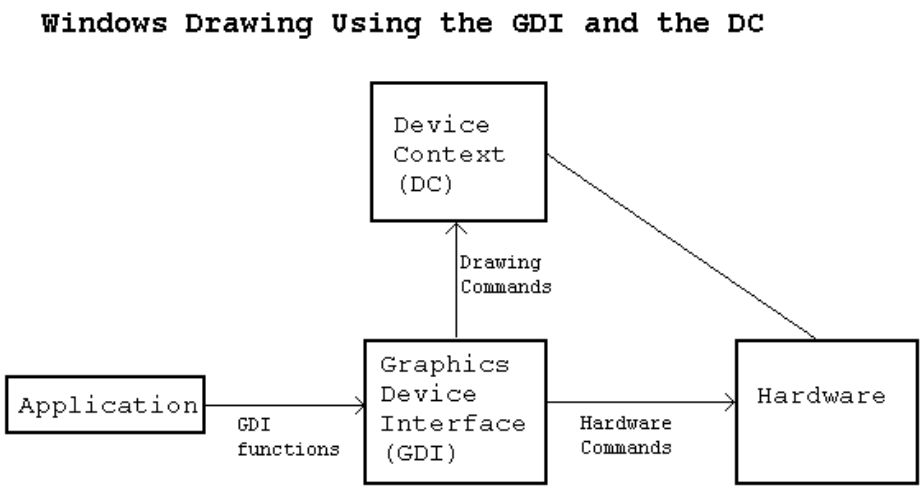
(C) Richard R. Eckert

Device Context

- Windows apps don't draw directly on hardware
- Draw on "Device Context" (DC)
  - Is associated with a physical device
  - Abstracts the device it represents
  - Like a painter's canvas
  - Must be "gotten" from Windows
  - Specifies drawing attribute settings
    - e.g., text color
  - Contains drawing objects
    - e.g., pens, brushes, bitmaps, fonts

(C) Richard R. Eckert

The DC and the GDI



(C) Richard R. Eckert

Some GDI Attribute Settings

ATTRIBUTE	DEFAULT	FUNCTION
Background color	white	SetBkColor()
Background mode	OPAQUE	SetBkMode()
Current Position	(0,0)	MoveToEx()
Drawing Mode	R2COPYPEN	SetROP2()
Mapping Mode	MM_TEXT	SetMapMode()
Text Color	Black	SetTextColor()

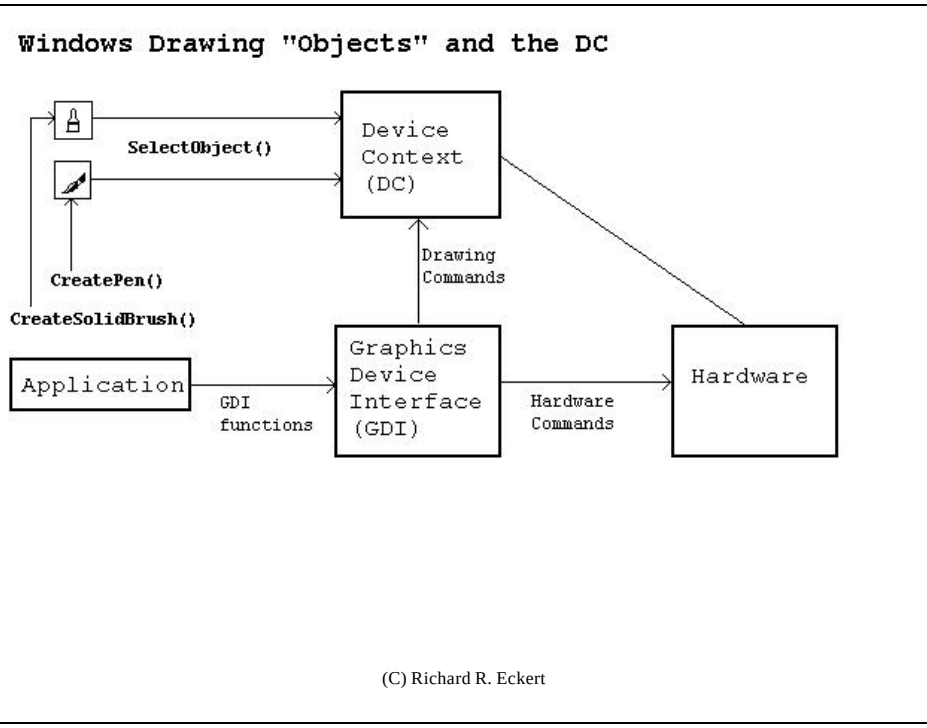
(C) Richard R. Eckert

Some GDI Drawing Objects

Object	Default	What it is
Bitmap	none	image object
Brush	WHITE_BRUSH	area fill object
Font	SYSTEM_FONT	text font object
Pen	BLACK_PEN	line-drawing object
Color Palette	DEFAULT_PALETTE	colors

- Can be created with GDI functions
- Must be "selected" into a DC to be used

(C) Richard R. Eckert



(C) Richard R. Eckert

Color in Windows

- Uses four-byte numbers to represent colors
- Simplest method--direct color:
  - typedef DWORD COLORREF;

```
 ------------------------------------------------------
| 0 |  Blue (0-255) | Green (0-255) | Red (0-255)  |
 ------------------------------------------------------
```

  - MSB=0:
    - ==> RGB color used (default)
    - other bytes specify R, G, B intensities

(C) Richard R. Eckert

RGB() Macro

- Specify Red, Green, Blue intensities
- RGB() generates a COLORREF value
- Can be used in color-setting functions), e.g.

```
COLORREF  cr;
cr = RGB (0,0,255);   /* bright blue */
```

- Example usage in a program

```
SetTextColor (hDC, RGB(255,0,0)); //red text
SetBkColor (hDC, RGB(0,0,255)); //blue bkgnd
```

(C) Richard R. Eckert

A Typical Sequence With Drawing Objects:

```
HPEN    hOldP, hNewP;
HDC     hDC;
hDC = GetDC(hWnd);
hNewP = CreatePen(PS_SOLID, 3, RGB(0,0,255));
hOldP = (HPEN)SelectObject(hDC, hNewP);
// NOW DO SOME DRAWING WITH THE NEW PEN
SelectObject(hDC,hOldP); //displace pen from DC
DeleteObject(hNewP); //now can be deleted
ReleaseDC(hWnd,hDC);
```

(C) Richard R. Eckert

Some GDI Drawing Primitives

- *Arc(hDC,x1,y1,x2,y2,xStart,yStart,xEnd,yEnd);*
- *Ellipse (hDc, x1,y1,x2,y2);*
- *MovetoEx (hDC,x1,y1,lpPoint);*
- *LineTo (hDC,x1,y1);*
- *Polygon (hDC,points_array,nCount);*
- *Polyline (hDC,points_array,nCount);*
- *Rectangle (hDC,x1,y1,x2,y2);*
- *SetPixel (hDC,x1,y1,colref);*
- *TextOut (hDC,x,y,lpString,cbString);*
- *Many more (see on-line help)*

(C) Richard R. Eckert

Stock Objects

- predefined in Windows
- obtain with GetStockObject();
  - gets a handle to a predefined pen/brush/font
- Stock objects are maintained by Windows
  - should not be deleted!
- Example

  SelectObject (hDC,
  GetStockObject(BLACK_PEN));

(C) Richard R. Eckert

Some Stock Objects

Some Object Choices:

Pen	BLACK_PEN, WHITE_PEN
Brush	DKGRAY_BRUSH, GRAY_BRUSH, BLACK_BRUSH, LTGRAY_BRUSH, NULL_BRUSH, WHITE_BRUSH
Font	ANSI_FIXED_FONT, ANSI_VAR_FONT, DEVICE_DEFAULT_FONT, SYSTEM_FONT, OEM_FIXED_FONT, SYSTEM_FIXED_FONT

(C) Richard R. Eckert

The winapp2.cpp Application

- Details of WndProc()--
  - menu item clicked==>WM_COMMAND message
    - LOWORD(wParam)==ID_RECTANGLE ("Rectangle" menu item clicked):
      - draw red-outlined rectangle (pen) with solid cyan interior (brush)
    - LOWORD(wParam)==ID_CIRCLE ("Circle" clicked):
      - draw blue-outlined circle (pen) with crosshatched magenta interior (brush)

(C) Richard R. Eckert

  - LOWORD(wParam)==ID_CLEAR ("Clear Screen" clicked):
    - call InvalidateRect() ==> Windows sends WM_PAINT message
      - client area needs to be repainted
      - default Window Procedure repaints client area with class background brush
      - effectively erases window's client area
  - LOWORD(wParam)==ID_QUIT ("Quit" clicked):
    - program calls DestroyWindow()
    - causes Windows to destroy window
    - and send a WM_DESTROY message
    - Handler posts WM_QUIT message
      - Which causes program to exit message loop and terminate

(C) Richard R. Eckert

- left mouse button pressed ==> WM_LBUTTONDOWN message
  - get cursor's x,y coordinates from lParam
    - use LOWORD & HIWORD macros
    - output "L" at (x,y) on screen DC with TextOut()
- right mouse button pressed ==> WM_RBUTTONDOWN message
    - output "R" at (x,y) on screen DC with TextOut()

(C) Richard R. Eckert

  - User hits ANSI character keyboard key/s ==> WM_CHAR message (wParam=char code)
    - copy character into a buffer
    - output buffer to upper left corner with TextOut()
  - User takes action to close window (double clicks on System menu or hits Alt-F4) ==> WM_DESTROY message
    - post WM_QUIT message to app's queue
    - causes program to exit event loop and return control to Windows

(C) Richard R. Eckert

Using Visual Studio to Create a Win32 API Application with a Menu and an Icon

1. Get into Visual Studio, open a New Project, and create an empty Win32 application
2. Create a new Visual C++ source file, type or paste in the code (winapp2.cpp), and save it as a C++ source file in the app's subdirectory
   - must have: #include "resource.h"
3. Add it to the project

(C) Richard R. Eckert

4. Create the Icon Resource (and the .rc file)
   - Select 'Project | Add Resource | Icon | New'
     - Brings up icon editor
   - Draw desired icon
   - Click on IDI_ICON1 in "Resource View" to bring up the "Properties" window and change the icon ID to "MYICON"
     - Don't forget the quote marks
   - Give a name to .ico file (or leave the default name)

(C) Richard R. Eckert

5. Select ‘Project | Add Resource | Menu | New’
   - Brings up the menu editor
     - Type the caption: &Circle in the “Type Here” rectangle
     - In resulting "Properties" box, Select “False” for “Pop-up”
     - Click on the resulting Circle menu item to bring up the “Properties” box again.
     - Note the default ID of ID_CIRCLE
   - Click on the next rectangle over in the menu editor
     - Repeat the above steps using caption: &Rectangle
     - Keep the default IDs
   - Repeat for: Clear &Screen, &Quit menu items
     - Keep default IDs

(C) Richard R. Eckert

6. Click on “IDI_MENU1” in “Resource View” to bring up the “Properties” window and change the menu ID to “MYMENU”
   - Don’t forget the quote marks
7. Build the project

(C) Richard R. Eckert

Copy Project to a Diskette or CD

- To save space, delete all the temporary files from the application's Debug Directories
  - Everything except the application itself (the .exe file)
- Also delete the VC++ Intellisense Database file from the topmost directory
- Copy the entire topmost directory to your diskette or CD-ROM
- If using a public computer, delete the workspace directory from the hard disk

(C) Richard R. Eckert