

Menus

Windows provides support for complex menus—
Popup menus
Menu bars that are graphics images
Enabled/disabled/grayed-out menu items
Checked/unchecked menu items
Menu items with associated bitmaps
Menu items that change dynamically as program runs
Good for pgms that operate in more than one state
Or to support beginner/advanced versions of menu

Creating Menus

- Can write source .RC resource script file containing menu definition
- Or use Developer Studio's menu editor to create menu visually

● Simple Menu Syntax—

```
MenuName MENU
BEGIN
    /* menu definition goes here */
END
```

Menu Syntax

- *MenuName*: string used to find menu data in program resources
- Menu Items
 - Go between BEGIN and END
 - Can only be MENUITEM or POPUP
- **Menu Item Syntax—**
MENUITEM string, MenuID options
or *MENUITEM SEPARATOR*
Latter Causes horizontal line between previous and following menu items

Menu Item Syntax

- *MENUITEM string, MenuID, Option*
 - *String*: Menu item's characters enclosed in " "
 - *MenuID*: Number passed as LOWORD(wParam) with WM_COMMAND msg
 - Usually given a constant name
 - *Option*:
 - Appearance: ENABLED, GRAYED, or INACTIVE
 - Check State: CHECKED, UNCHECKED
 - Refers to check mark next to menu item

Menu Item Syntax

- **Popup menus**
 - Used when number of menu items gets too big
 - Can have nested popups (up to 8 levels)
- **Popup syntax**
 - *POPUP string options*
 - *string*:
 - Gives popup title--what will appear on menu bar
 - No ID needed since popup titles not selectable & don't generate messages
 - *Some options*:
MENUBARBREAK
MENUBREAK

Changing Menu Item Status

1. Get handle to entire menu *GetMenu(hWnd)*
 - Returns handle to menu attached to specified window
2. Change Status (activate/deactivate an item)
 - *EnableMenuItem (hMenu, idEnableItem, ActionFlag);*
 - hMenu=handle to menu containing item
 - idEnableItem: which item
 - ActionFlag: how & what action
 - Examples:
 1. MF_BYCOMMAND | MF_ENABLED → Enable menu item whose ID is given in 2nd parm
 2. MF_BYPOSITION | MF_DISABLED → Disable menu item whose position given in 2nd parm
 - Position number relative to top left item (position 0)
 - Hard to keep track of positions, so not used often

Examples

- 1. `EnableMenuItem(hMenu, IDM_SEL3, MF_BYCOMMAND / MF_ENABLED);`
- 2. `EnableMenuItem(hMenu, 5, MF_BYPOSITION / MF_GRAYED);`
- Possible actions:
 - MF_ENABLED
 - MF_DISABLED (seldom used, since confusing to user)
 - MF_GRAYED

Changing Check State

- `CheckMenuItem()`
 - Checks/uncheckeds specified item
 - Works like `EnableMenuItem()`
 - Action flag values:
 - MF_CHECKED or MF_UNCHECKED
 - Can use bitmaps for checked/unchecked state
 - SetMenuItemBitmaps (hMenu, pos'n, action flags, h_unchecked bitmap, h_checked bitmap);
 - Action flags:
 - MF_BYCOMMAND or MF_BYPOSITION

Getting Menu Item State

`GetMenuItemState (hMenu, menu_id, MF_Flags)`

- Returns `UINT` that encodes menu item status
- A combination of `MF_CHECKED`, `MF_ENABLED`, etc.

Creating Dynamic Menus (on fly as program operates)

- Rationale:
 - Operations may become impossible or irrelevant, so delete them from menu
 - Other operations may become possible or relevant, so add them to menu
 - May want to use bitmap images as menu items
 - e.g., tool selection (picking a brush image for painting)
 - Graphical menu items can't be defined in resource script
- Can be created as the program runs

Menu-altering Functions

- `CreateMenu();` Creates new menu, ready to add items
- `CreatePopupMenu();` Creates new popup menu, ready to receive items
- `SetMenu();` Attaches a menu to a window
 - Often used with `LoadMenu()` to switch between alt. menus
- `AppendMenu();` Adds new menu item or popup to end of a menu
- `InsertMenu();` Inserts new menu item/popup into a menu/popup menu
- `DeleteMenu();` Removes menu item from a menu or popup menu

- `DestroyMenu();` Deletes an entire menu, removing it from memory
 - Only needed if menu was loaded but not attached to a window
- `DrawMenuBar();` Draws the menu bar (in menu area below window caption)
 - Makes any changes visible
- `LoadMenu();` Loads menu from program's resource data
 - Ready to be attached to a Window with `SetMenu()`

Basic sequence

1. **CreateMenu()**: Create a new, empty menu
 - Returns a handle to the new menu
2. **AppendMenu()** and/or **InsertMenu()**
 - Add menu items as needed
3. **SetMenu()**: Attach menu to a window
 - **Popup menus** must be created separately and attached to menu as follows:
 1. **CreatePopupMenu()**: Create a new, empty popup menu
 - Returns a handle to the new popup menu
 2. **AppendMenu()** or **InsertMenu()**: Add menu items to popup
 3. **AppendMenu()** or **InsertMenu()**: Add popup to main menu

Appending Item at End of Menu

AppendMenu (hMenu, MF_flags, item_id, item_content);

- hMenu: which menu to append item to
- MF_flags, Bitwise OR of:
 - What: MF_BITMAP, MF_STRING, MF_POPOP
 - Appearance: MF_ENABLED, MF_GRAYED, etc.
- item_id: from resource data (IDMs) or hPopup
- item_content: what goes there: the string or hBitmap
 - Example: "&Quit", (LPSTR)hImage

Inserting a Menu Item in any Position

InsertMenu (hMenu, item_id, MF_flags, new_item_id, item_content);

- item_id: where (in front of this item)
 - position or IDM_***
- MF_flags, Bitwise OR of:
 - where spec.: MF_BYCOMMAND, MF_BYPOSITION
 - What: MF_BITMAP, MF_STRING, MF_POPOP
 - Appearance: MF_ENABLED, MF_GRAYED, etc.
- new_item_id: IDM_*** or hPopup
- item_content: what goes there; the string or hBitmap

Deleting a Menu Item

DeleteMenu (hMenu, item_id, MF_flags)

DestroyMenu()

- Must destroy unattached menus
 - If not, they will remain in memory for entire Windows session
 - Attached menus are destroyed automatically when window is destroyed

InsertMenu() or DeleteMenu()

- Can be used to change existing menus
 - Usually easier than creating an entire menu from scratch
 - More flexible than defining multiple menus in program's resources and switching between them with **LoadMenu()** and **SetMenu()**

Creating a menu with bitmap images

1. Create image as bitmap (.bmp) using Dev. Studio
2. Include bitmap in program's resource data
3. Use *LoadBitmap()* to get bitmap data while program is running
 - Returns a handle to the bitmap
4. Use *AppendMenu()* or *InsertMenu()* to add the bitmap as a menu item
5. Use *SetMenu()* to attach the menu to the window
6. At termination use *DeleteObject()* to remove bitmap from memory

MENU2 Example Program

- Example of dynamic menus
- No menu defined in .rc file
- Main menu created upon receipt of the WM_CREATE message
- “Tools” popup menu has three bitmap images
 - Clicking on each changes mouse cursor to that shape →
 - Need to include two cursors (third is predefined ARROW cursor)
 - And the three bitmaps in resource script file
 - (Bitmaps & cursors have different formats, need both)

● “Add Menu Items”

- Adds a new popup menu w/ items:
 - 1. “New Selection 1” toggles its check state & activation state of following item
 - 2. Item to be toggled by Selection 1; if active, causes a beep
 - 3. Delete new popup menu (& reactivate old item)
- Also Grays out old “Add Menu Items” item

MENU2 Resources using Visual Editors—

- **Cursors:**
 - ID="CUTCURSOR", filename: cutcur.cur
 - ID="GLUECUR", filename: gluecur.cur
 - Create/insert into project with Cursor Editors
- **Bitmaps:**
 - ID="CUTBMP", filename: cutbmp.bmp
 - ID="PASTEBMP", filename: pastebmp.bmp
 - ID="ARROWBMP", filename: arrowbmp.bmp
 - Use Bitmap Editor to create bitmap resources
 - Same way as Cursor Editor is used

Menu Resource

- None (since menu created dynamically in program)
- But still must assign constant values to menu item names (IDM_*)
- Done in the menu2.h
- Must be included along with resource.h

Constants

- ARROWCURSOR, GLUECURSOR, & CUTCURSOR
- Used in switch/case statement in program
- Constant values assigned in menu2.h file

The MENU2.CPP Program

- WM_CREATE: Create initial main menu
 - Create main menu and popup menu (empty); i.e., get handles
 - Load bitmaps to go into the popup menu
 - Append bitmaps to popup menu and items to main menu:
 - Attach entire menu structure to program's window with *SetMenu()*

Other menu items (WM_COMMAND)

- Create, add, delete new popup menu and items:
 - Use calls to *CreateMenu()*, *CreatePopupMenu()*, *AppendMenu()*, *InsertMenu()*, *DeleteMenu()*
- To change state of menu items
 - Use calls to *EnableMenuItem()* and *CheckMenuItem()*

Cursors

- User chooses bitmap from “Tools” popup
 - change nCursor variable that keeps track of current cursor
- User moves mouse in window (WM_SETCURSOR)
 - Examine nCursor & use *LoadCursor()* to get current mouse cursor
 - Use *SetCursor()* to change to current cursor

Other Stuff in MENU2

- Since menu is loaded dynamically, original menu when window class was registered is NULL
- When window is destroyed (WM_DESTROY), call *DeleteObject()* to get rid of the bitmaps