Windows Programming

Lecture 11

Windows Styles

- **WS_CAPTION** Creates a window that has a title bar (implies the **WS_BORDER** style).
- **WS_CHILD** Creates a child window. Cannot be used with the **WS_POPUP** style.
- **WS_DISABLED** Creates a window that is initially disabled.
- **WS_HSCROLL** Creates a window that has a horizontal scroll bar.
- **WS_MAXIMIZE** Creates a window of maximum size.
- **WS_VISIBLE** Creates a window that is initially visible.

Windows Styles

- **WS_MINIMIZE** Creates a window that is initially minimized. For use with the **WS_OVERLAPPED** style only.
- **WS_OVERLAPPED** Creates an overlapped window. An overlapped window usually has a caption and a border
- **WS_POPUP** Creates a pop-up window. Cannot be used with the **WS_CHILD** style.
- **WS_SYSMENU** Creates a window that has a Control-menu box in its title bar. Used only for windows with title bars
- **WS_VSCROLL** Creates a window that has a vertical scroll bar

Threads

- A thread is basically a path of execution through a program. It is also the smallest unit of execution that Win32 schedules. A thread consists of a stack, the state of the CPU registers, and an entry in the execution list of the system scheduler. Each thread shares all of the process's resources

- could be performed at the same time. Threads are an operating system feature that lets application logic be separated into several concurrent execution paths.

Threads

Threads allow complex applications to make more effective use of a CPU even on computers with a single CPU. With one CPU, only one thread can execute at a time. If one thread executes a long running operation that does not use the CPU, such as a disk read or write, then another one of the threads can execute until the first operation completes. Being able to execute threads while other threads are waiting for an operation to complete allows the application to maximize its use of the CPU. This is especially true for multiuser, disk I/O intensive applications such as a database server.

Worker Thread

A worker thread is commonly used to handle background tasks that the user shouldn’t have to wait for to continue using your application. Tasks such as recalculation and background printing are good examples of worker threads. This article details the steps necessary to create a worker thread.

User-interface Threads

In Windows, a thread that handles user input and responds to user events independently of threads executing other portions of the application. User-interface thread own one or more windows and have their own message queue. User-interface thread process messages received from the system.

Hierarchy of windows

The basic building block for displaying information in the Microsoft® Windows™ graphical environment is the window. Microsoft Windows manages how each window relates to all other windows in terms of visibility, ownership, and parent/child relationship. Windows uses this relationship information when creating, destroying, or displaying a window

Desktop Window

When Windows initializes, it creates a window, known as the *desktop window*, that is sized to cover the entire display area and upon which all other windows are displayed. The window manager uses the desktop window as the top of the window manager's list. Thus, the desktop window is the top of the window hierarchy.

Top-level Window

Windows that form the next level of the window hierarchy are called *top-level windows*. A top-level window is any window that is not a child window. Top-level windows do not have the WS_CHILD style. The window manager connects top-level windows to the desktop window by connecting the top-level windows into a linked list with the head of the link list stored in the desktop window's child window handle, and by using each top-level window's next sibling handle to form the list. The window manager places top-level windows at the top of the Z order when they are created, and thus the entire window is visible.

Top-level Window

Another type of relationship can exist between top-level windows: Top-level windows can own or be owned by other top-level windows. An owned window is always above its owner in the Z order and is hidden when its owner is minimized

Overlapped Window

- A window with the WS_OVERLAPPED style. Overlapped windows are top-level windows. These are designed to serve as an application's main window

- Overlapped window can only be owned by overlapped window.

- When owner window is minimized, owned window is hidden.
- Destroying owner window destroy sall owned windows.
- Owned windows are always displayed on the top of their owner window.

Popup Window

- Popup are nearly identical to overlapped except that pop up window may or may not have a caption; while overlapped window always has a titlebar/caption

- Popups are used for dialogs normally.

- In CreateWindow(), Overlapped and Popup windows have owners NOT parents.

Child window

The desktop window occupies the first level of the windows hierarchy and top-level windows occupy the second level. *Child windows*, which are windows created with the WS_CHILD style, occupy all other levels. The window manager connects child windows to their parent window in the same way it connects top-level windows to the desktop window.

Child window

- Child window is confined to parent's client area
- Only child can have a parent (in **CreateWindow()**)
- It can't have a menu, but may have a caption
- It overlays the parent, receives all messages in that area before parent does.
- Notification messages are sent to Parent Window bt Child Window.
- Child windows used for controls

A window may further contain more windows
e.g. In calculator: screen, radio buttons for deg, rad; checkboxes, pushbuttons showing 1, 2, 3, etc. ALL ...

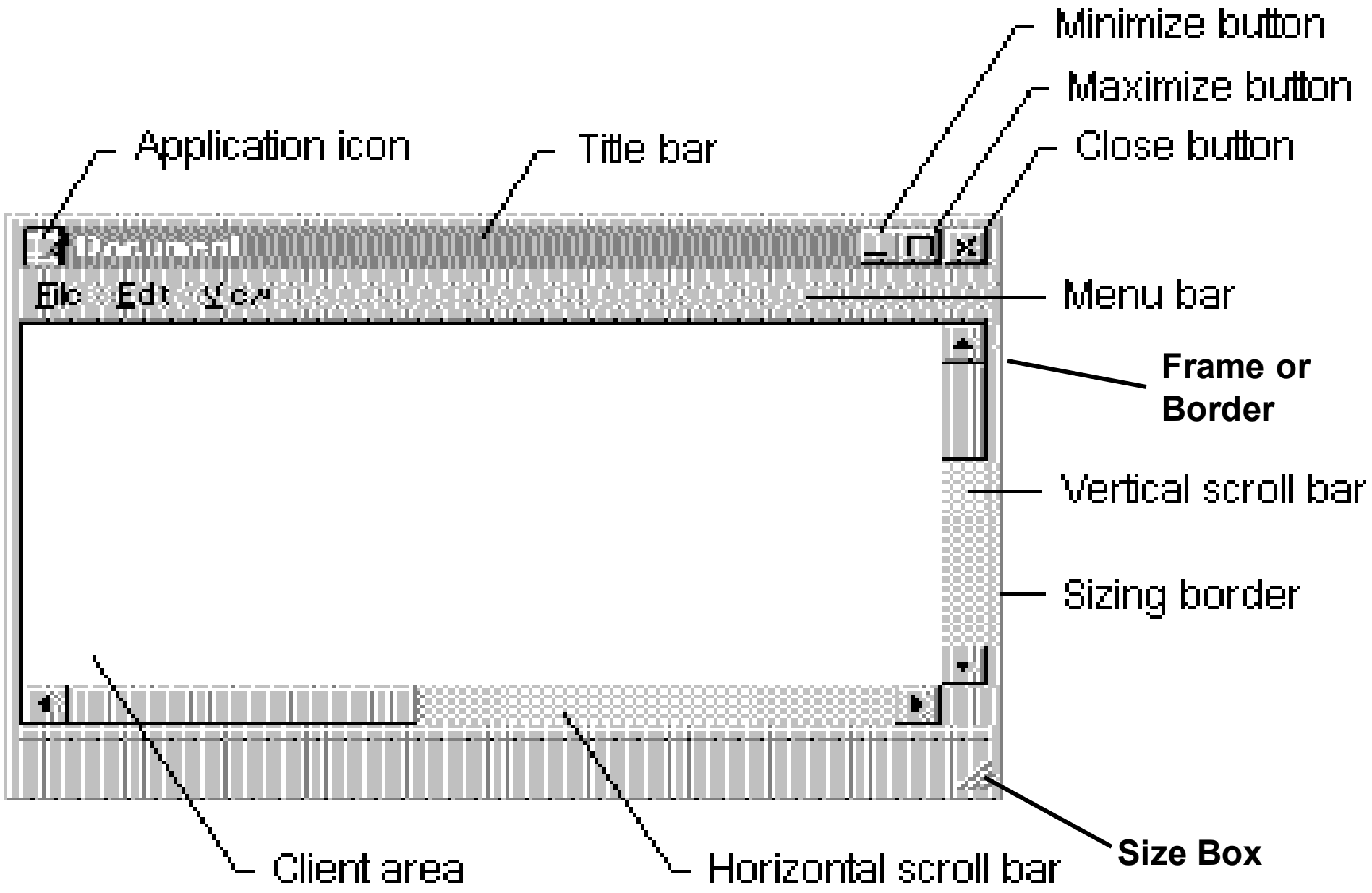
Controls

A control is a child window an application uses in conjunction with another window to perform simple input and output (I/O) tasks. Controls are most often used within dialog boxes, but they can also be used in other windows. Controls within dialog boxes provide the user with the means to type text, choose options, and direct a dialog box to complete its action. Controls in other windows provide a variety of services, such as letting the user choose commands, view status, and view and edit text.

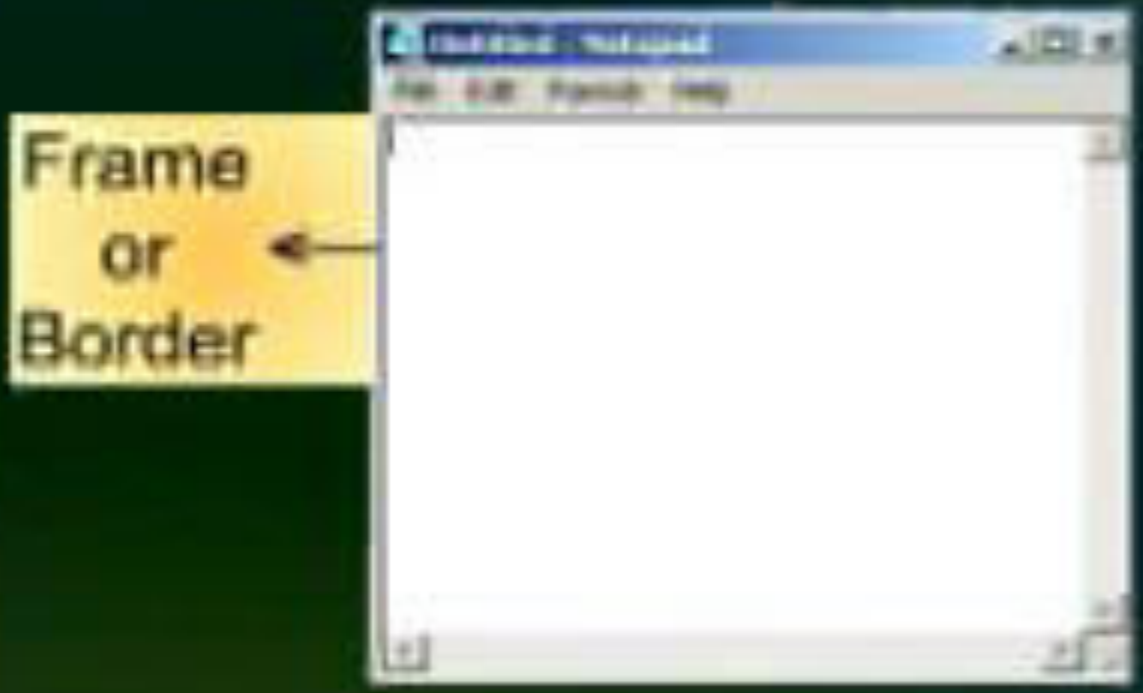
Controls

- Windows provides many pre-coded window classes for use as child windows called ***controls***

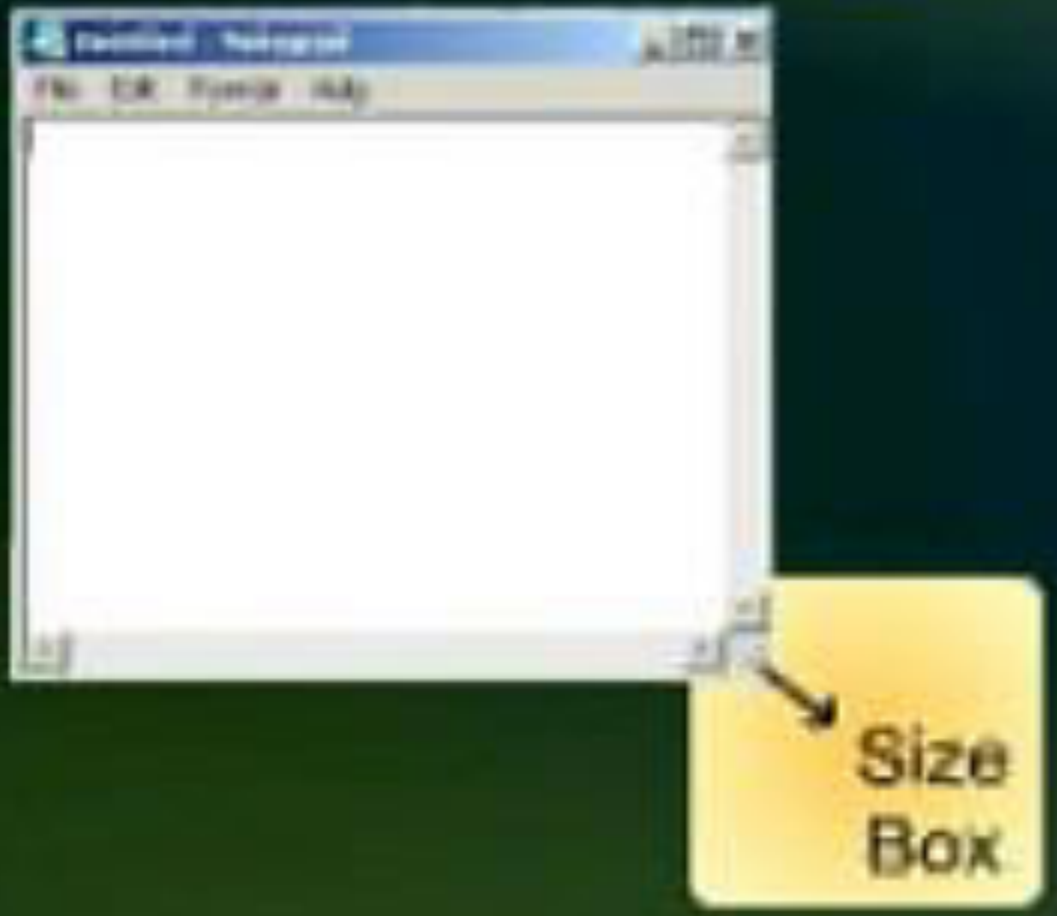
- Controls are ***child windows*** used for input/output purposes


Minimize button
Maximize button
Application icon
Title bar
Close button
Menu bar
Frame or Border
Vertical scroll bar
Sizing border
Client area
Horizontal scroll bar
Size Box

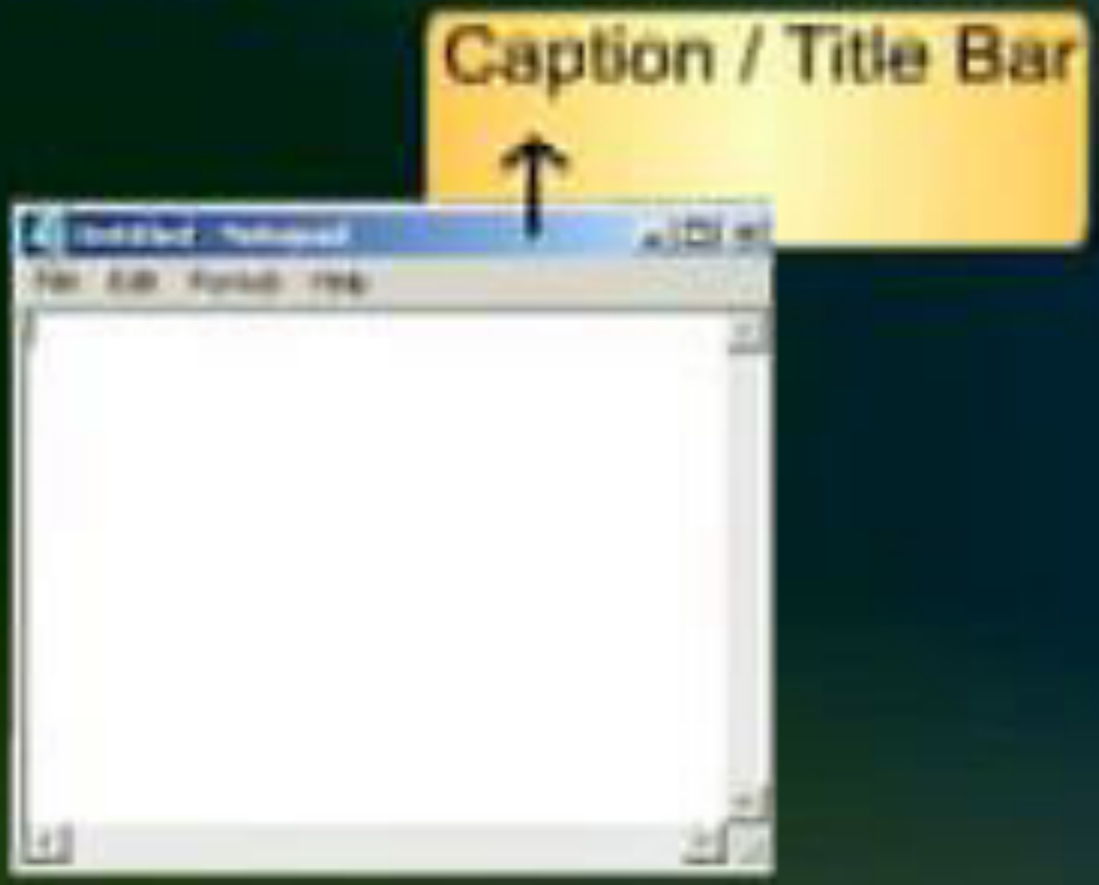
Parts of a Window



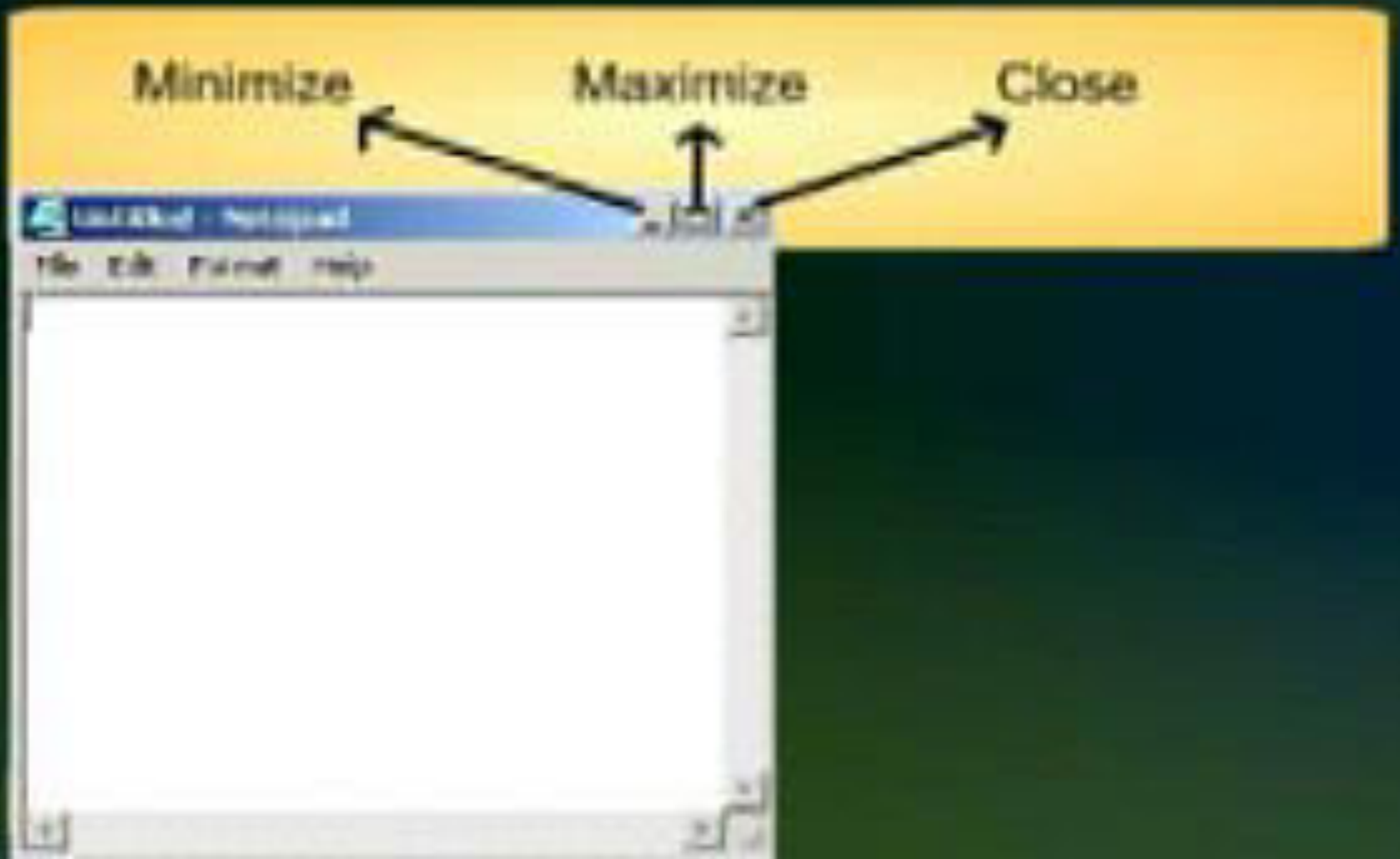
Parts of a Window



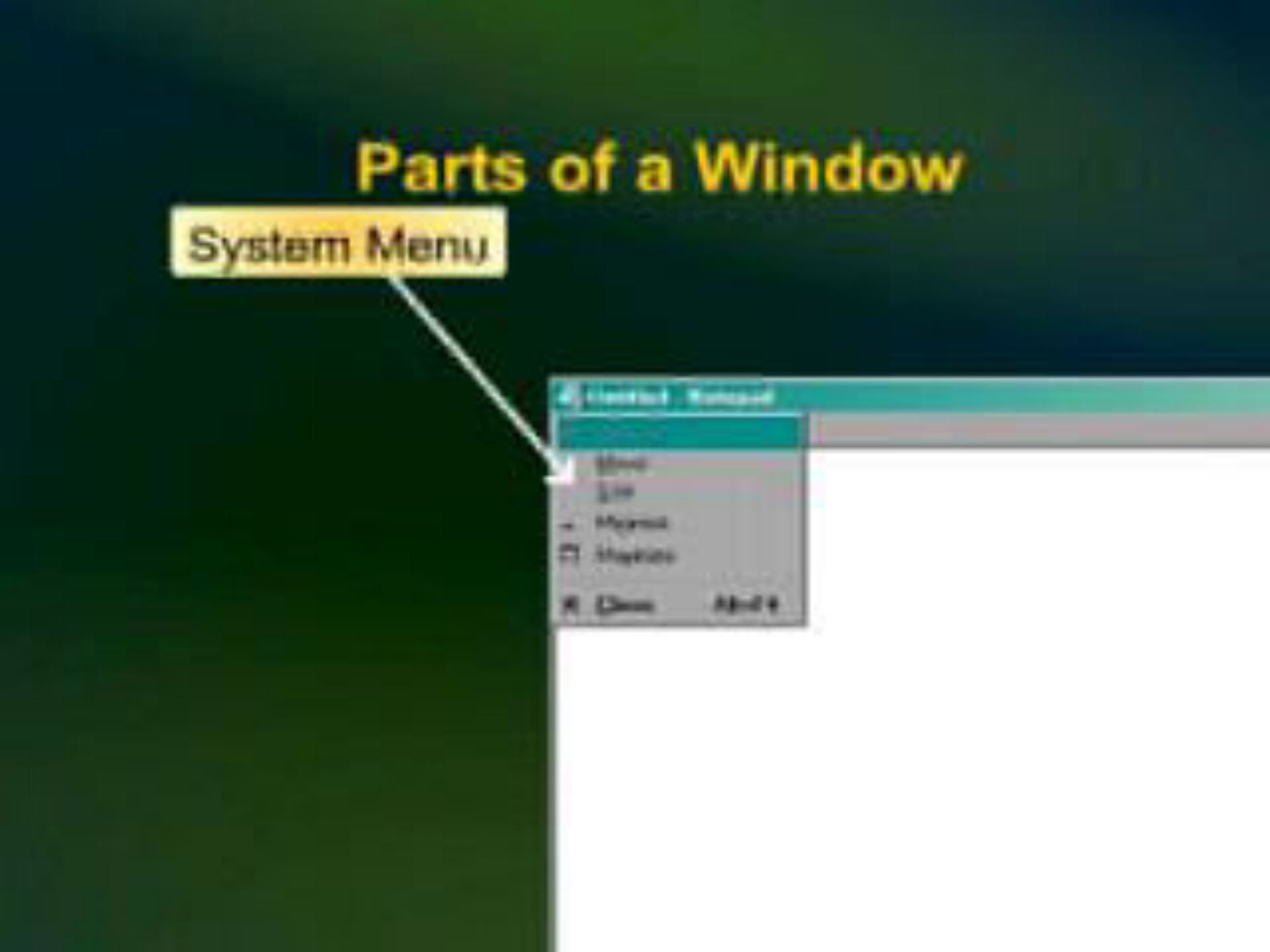
Parts of a Window



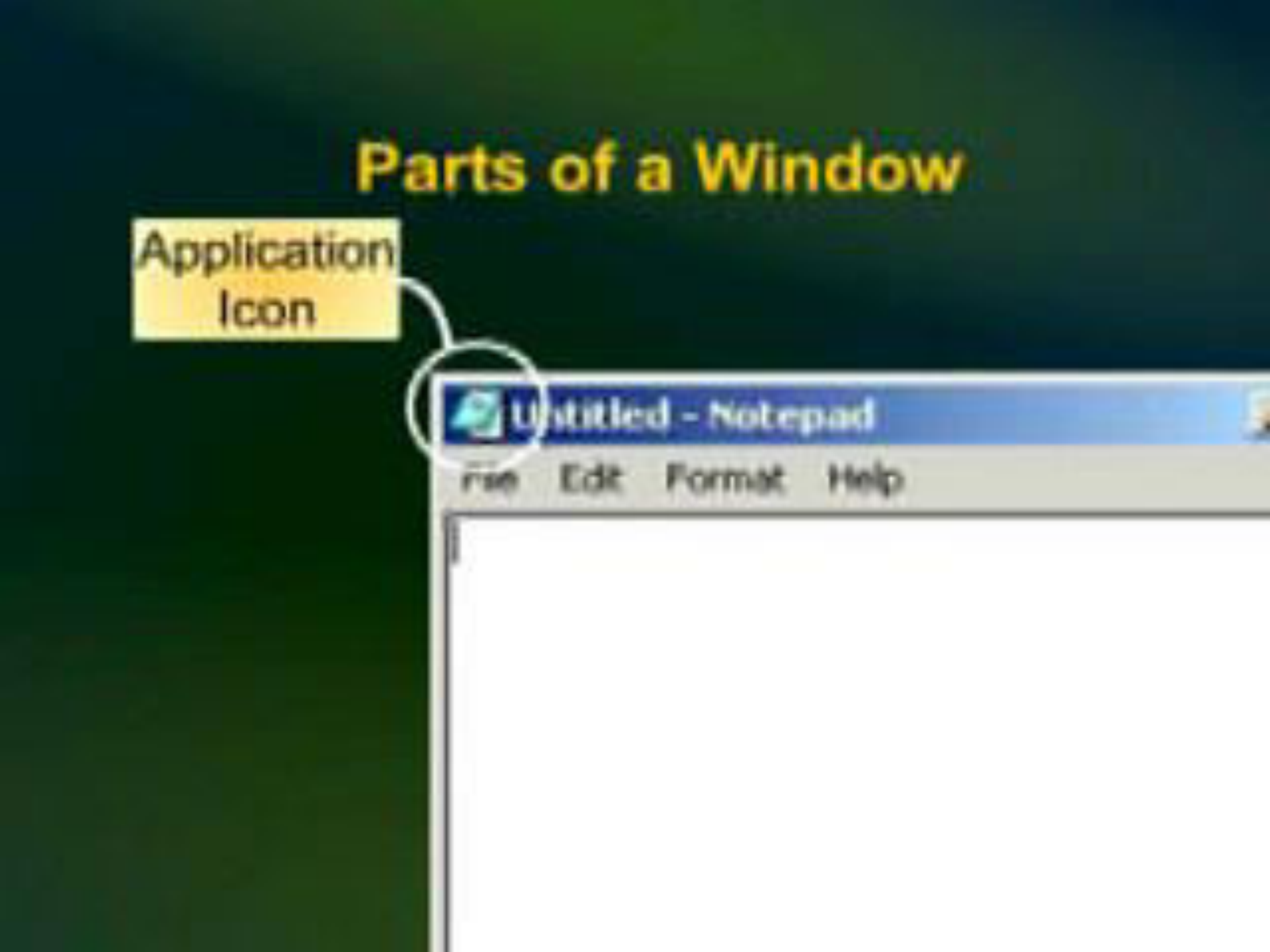
Parts of a Window



Parts of a Window



Parts of a Window

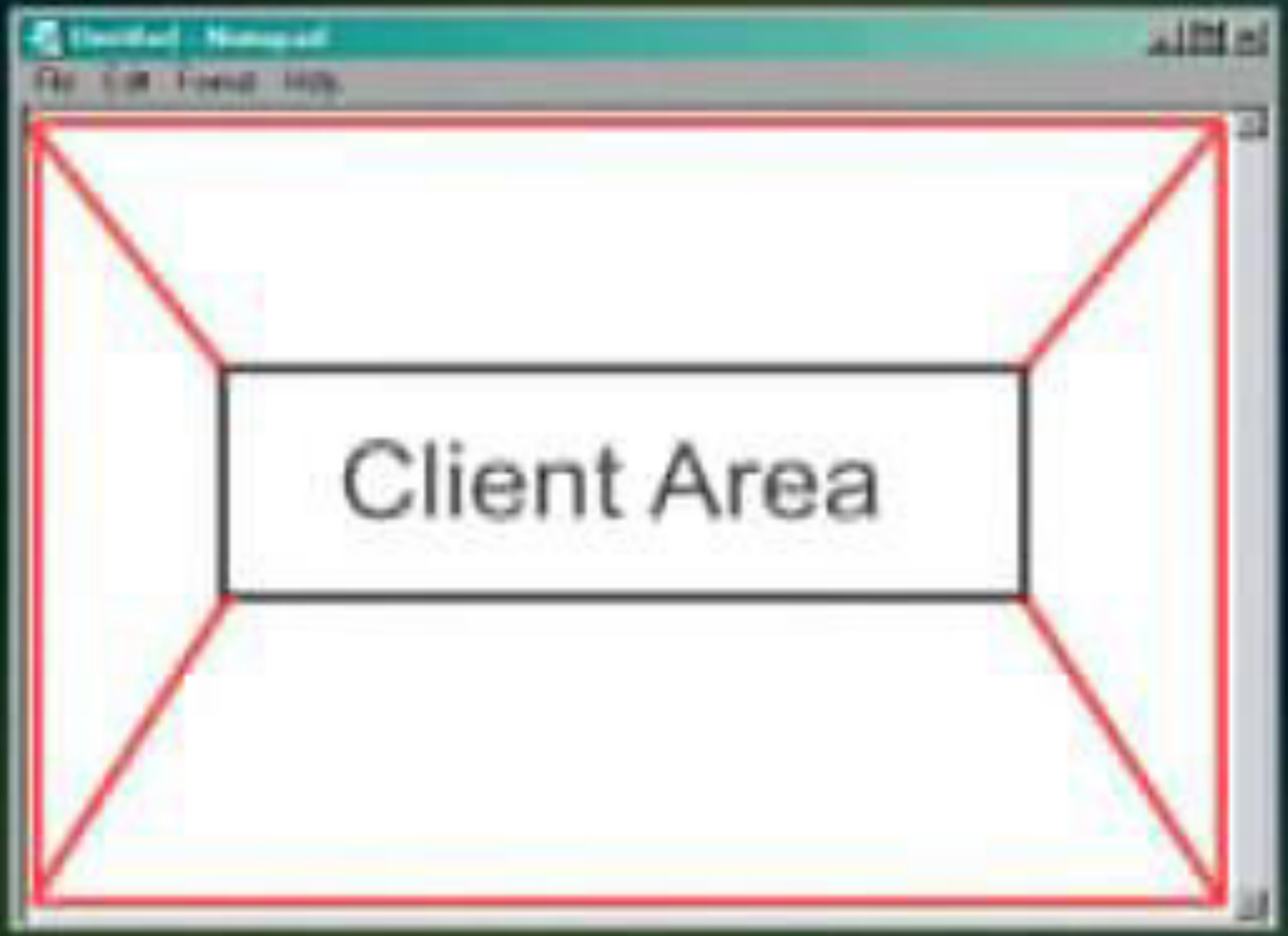


Parts of a Window

Client area and Non client area

- The portion of a window in which an application displays output, such as text or graphics is *called* client rectangle.

- The parts of a window that are not a part of the client area. A window's nonclient area consists of the border, menu bar, title bar, and scroll bar.

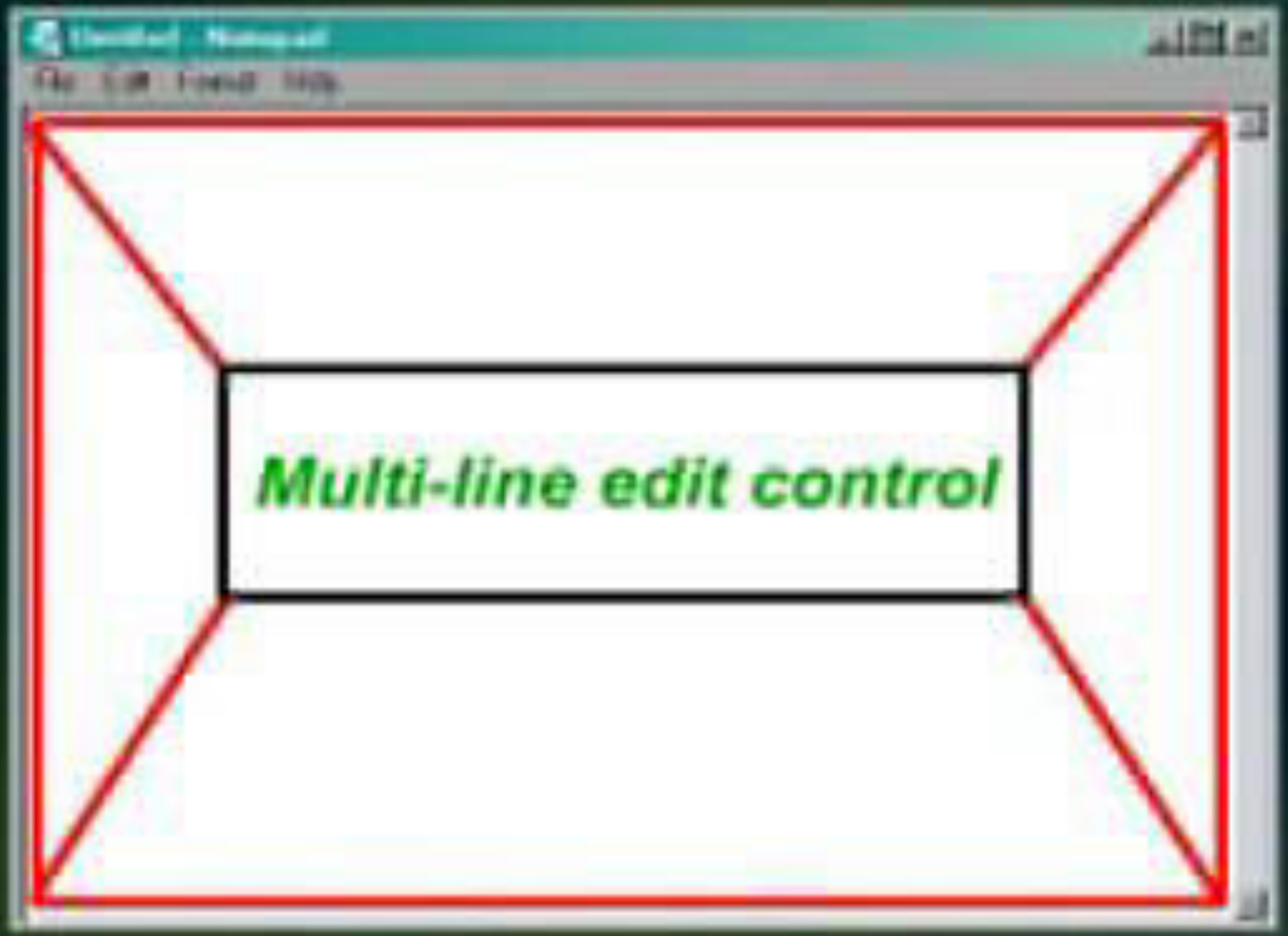

Client Area

1. Edit Control

An edit control is selected and receives the input focus when a user clicks the mouse inside it or presses the TAB key. After it is selected, the edit control displays its text (if any) and a flashing caret that indicates the insertion point. The user can then enter text, move the insertion point, or select text to be edited by using the keyboard or the mouse. An edit control can send notification messages to its parent window in the form of **WM_COMMAND** messages.

Controls


Calculator
Edit View Help
Single-line edit control
0
Backspace
CE
C
MC
7
8
9
/
sqrt
MR
4
5
6
*
%

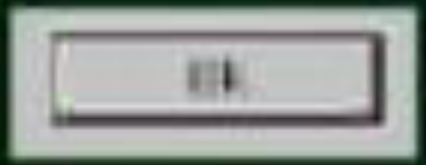

Multi-line edit control

Edit Control Styles

- **ES_MULTILINE**
- **ES_LEFT**
- **ES_CENTER**
- **ES_RIGHT**
- **ES_LOWERCASE**
- **ES_UPPERCASE**
- **ES_OEMCONVERT**

… …. ….

Controls
(*Buttons*)



Push Button

Check Box

Radio Button

Static Control

- A *static control* is a control that enables an application to provide the user with informational text and graphics that typically require no response.

- Applications often use static controls to label other controls or to separate a group of controls. Although static controls are child windows, they cannot be selected. Therefore, they cannot receive the keyboard focus

Controls

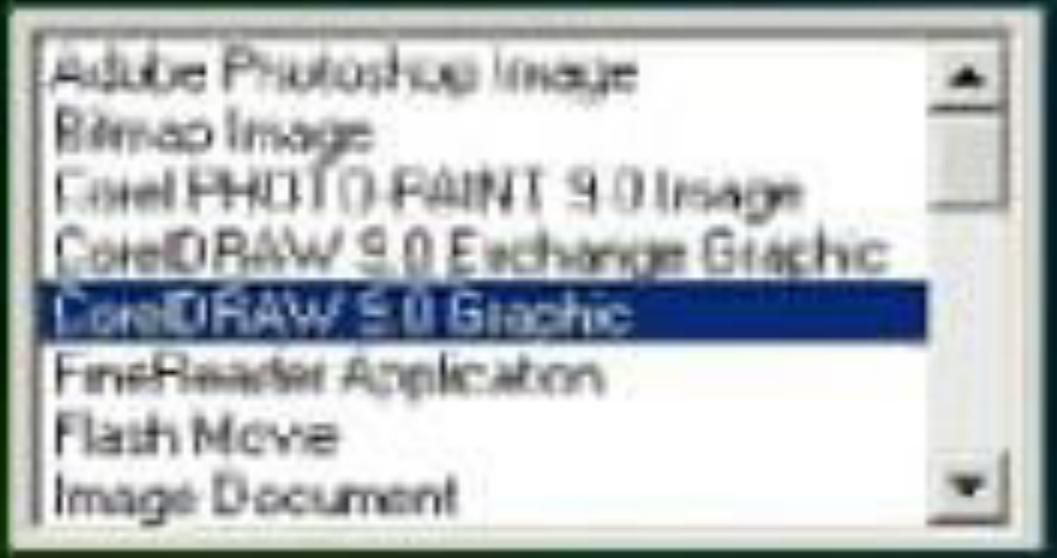
Static Control is used to display static text and labels in windows and dialogs

List box

list box is a control window that contains a list of items from which the user can choose. List box items can be represented by text strings, bitmaps, or both. If the list box is not large enough to display all the list box items at once, the list box provides a scroll bar. The user scrolls through the list box items, and applies or removes selection status as necessary. Selecting a list box item changes its visual appearance, usually by changing the text and background colors to those specified by the relevant operating system metrics. When the user selects or deselects an item, the system sends a notification message to the parent window of the list box.

Controls

List Box



Combo Box

A *combo box* is a unique type of control, defined by the COMBOBOX class, that combines much of the functionality of a list box and an edit control.

The Win32® API provides three types of combo boxes:

- Simple combo boxes (CBS_SIMPLE)
- Drop-down combo boxes (CBS_DROPDOWN)
- Drop-down list boxes (CBS_DROPDOWNLIST

Controls

Combo Box

Combo Box

The Win32® API provides three types of combo boxes:

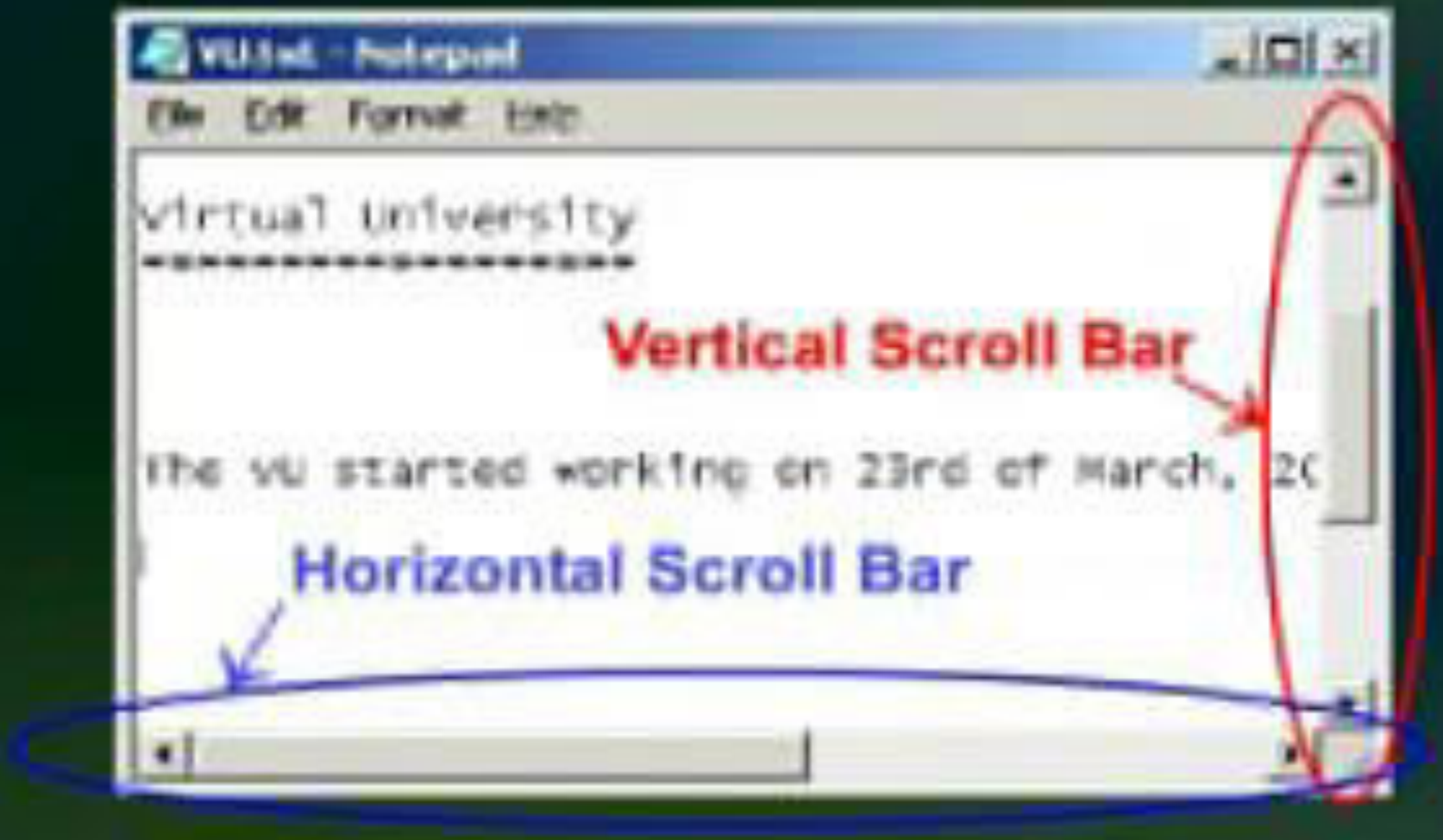
- Simple combo boxes (CBS_SIMPLE)
- Drop-down combo boxes (CBS_DROPDOWN)
- Drop-down list boxes (CBS_DROPDOWNLIST

A combo box consists of a list and a selection field. The list presents the options that a user can select and the selection field displays the current selection. Except in drop-down list boxes, the selection field is an edit control and can be used to enter text not available in the list.

Scroll Bar

A window in an Win32®-based application can display a data object, such as a document or a bitmap, that is larger than the window's client area. When provided with a *scroll bar*, the user can scroll a data object in the client area to bring into view the portions of the object that extend beyond the borders of the window.

Controls

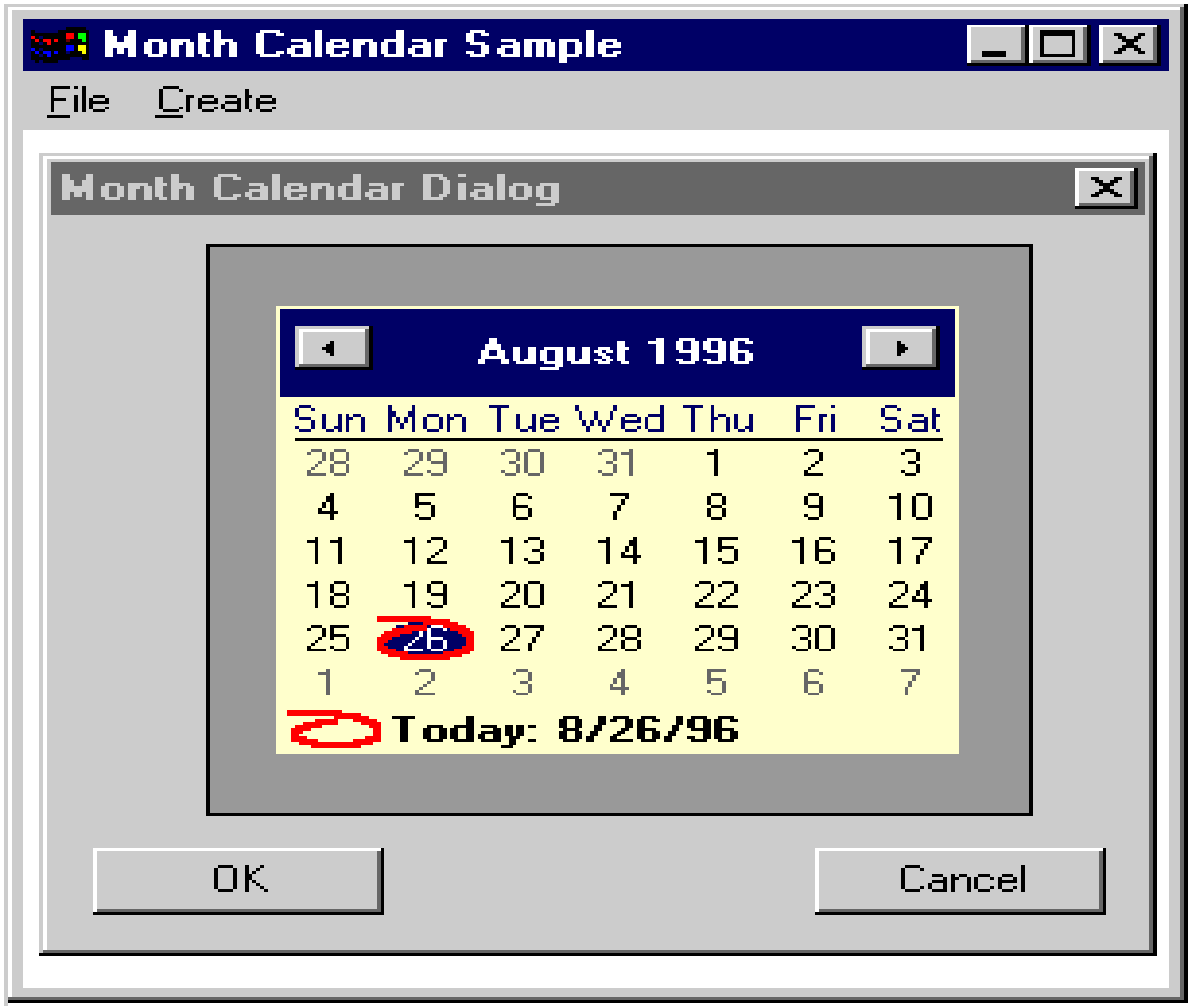

File Edit Format
virtual university
Vertical Scroll Bar
the vu started working on 23rd of march,
Horizontal Scroll Bar

Common Controls

The common controls are a set of windows that are implemented by the common control library, which is a dynamic-link library (DLL) included with the Microsoft® Windows® operating system. Like other control windows, a common control is a child window that an application uses in conjunction with another window to perform I/O tasks.

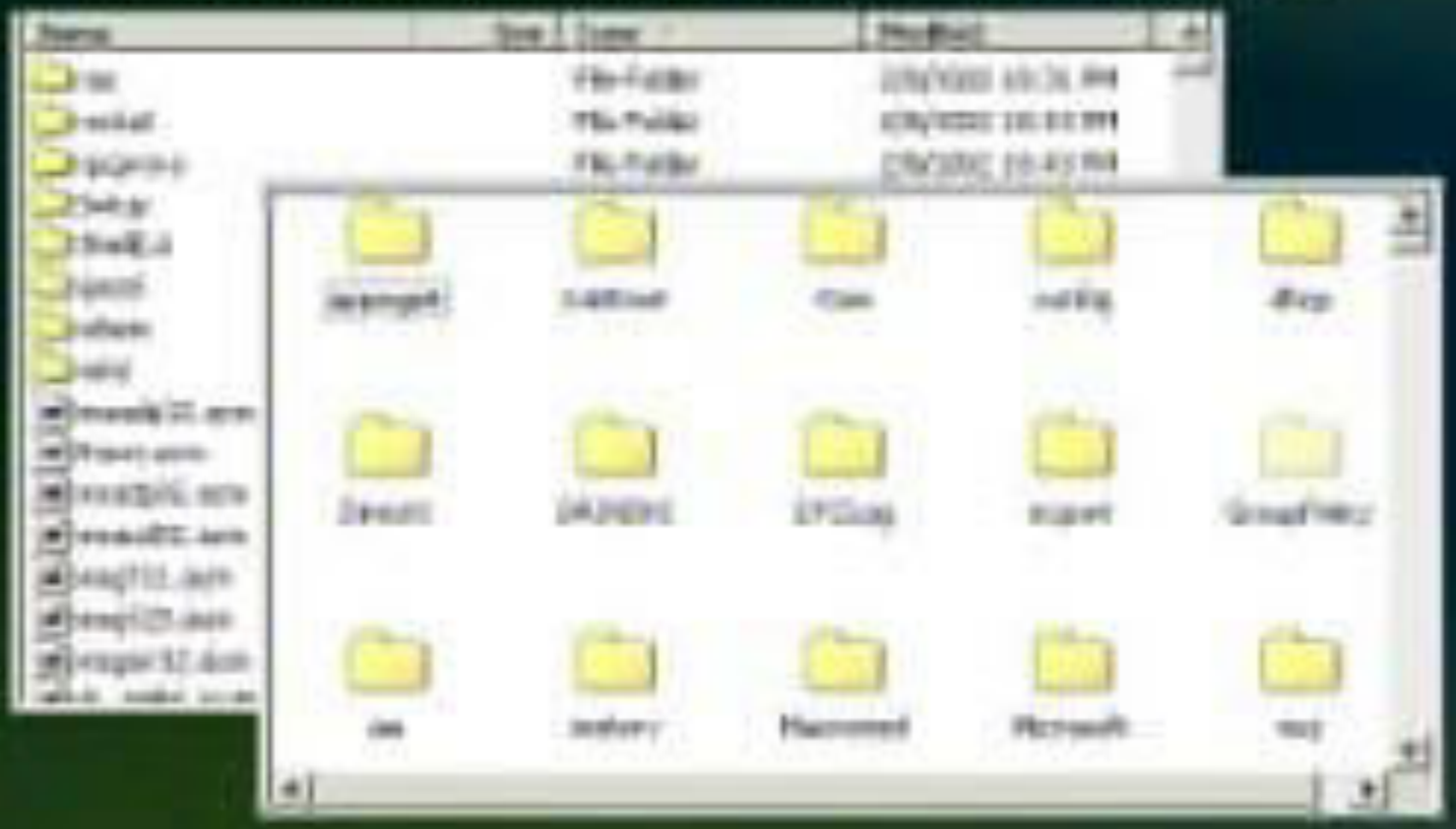
Date Time Picker

a common Control



Common Controls

List View (*in report and icon styles*)



Hotkey Control

Only shortcut keys (hotkeys) can be typed in.

Other User Interface Elements

- Cursors
- Icons
- Bitmaps (can add .bmp pictures)
- Accelerators (e.g CTRL + S)

Common messages (brief description)

- **WM_ACTIVATE**
- **WM_CLOSE**
- **WM_CREATE**
- **WM_ENABLE**
- **WM_LBUTTONDOWN**
- **WM_PAINT**
- **WM_RBUTTONDOWN**
- **WM_QUIT**
- **WM_SETTEXT**
- **WM_COMMAND**
- **WM_DESTROY**
- **WM_SYSCOMMAND**

WM_SYSCOMMAND

A window receives this message when the user chooses a command from the **window** menu (formerly known as the system or control menu) or when the user chooses the maximize button, minimize button, restore button, or close button.

WM_SYSCOMMAND

- *wParam*

  Specifies the type of system command requested. This parameter can be one of the following values

- *lParam*

  The low-order word specifies the horizontal position of the cursor, in screen coordinates, if a window menu command is chosen with the mouse. Otherwise, this parameter is not used. The high-order word specifies the vertical position of the cursor, in screen coordinates, if a window menu command is chosen with the mouse. This parameter is –1 if the command is chosen using a system accelerator, or zero if using a mnemonic.

Swap minimize/maximize

- Many applications ask: “Are you sure you want to quit?” e.g. in Notepad if the file is not saved. Our application will do the same using the **MessageBox()** API function.

- Applicaiton will have the functions of maximise/minimise buttons swapped.

The window procedure (*switch only*)

```
case WM_SYSCOMMAND:
        wParam &= 0xFFF0;  // lower 4-bits used by system
        switch(wParam)
        {
        case SC_MAXIMIZE:
                wParam = SC_MINIMIZE;
                return DefWindowProc(hWnd, message, wParam, lParam);

        case SC_MINIMIZE:
                wParam = SC_MAXIMIZE;
                return DefWindowProc(hWnd, message, wParam, lParam);

        case SC_CLOSE:
                if(MessageBox(hWnd, "Are you sure to quit?",
                                "Please Confirm", MB_YESNO) == IDYES)
                        DestroyWindow(hWnd);
                break;
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
                break;
        }
        break;
    case WM_DESTROY:   PostQuitMessage(0);          break;
    default:   return DefWindowProc(hWnd, message, wParam, lParam);
```

WM_SYSCOMMAND and WM_CLOSE

- Close message is quite often programmed like this with a quit confirmation.
- If we don't handle **SC_CLOSE** case and pass it to **DefWindowProc()**, it sends another message to window procedure. YES, it is possible to manually send a message to a window.
- **WM_CLOSE** message is sent after this, so our new window procedure may look like this

The window procedure (*switch only*)

```
case WM_CLOSE: if(MessageBox(hWnd, "Are you sure to quit?",
"Please Confirm", MB_YESNO) == IDYES) DestroyWindow(hWnd); break;
case WM_SYSCOMMAND:
    wParam &= 0xFFF0;  // lower 4-bits used by system
    switch(wParam)
    {
    case SC_MAXIMIZE:
           wParam = SC_MINIMIZE;
           return DefWindowProc(hWnd, message, wParam, lParam);

    case SC_MINIMIZE:
           wParam = SC_MAXIMIZE;
           return DefWindowProc(hWnd, message, wParam, lParam);

    default:
           return DefWindowProc(hWnd, message, wParam, lParam);
           break;
    }
    break;
case WM_DESTROY:   PostQuitMessage(0);            break;
default:   return DefWindowProc(hWnd, message, wParam, lParam);
```

User-friendly applications

- Our app was non-standard
- Standard interface functions close, maximize, minimize etc. should have the same meaning
- We ask every time “Are you sure..”, it’s annoying. Must be user-friendly.