Latest updated by: HanhTT1

Agenda

- Menu and Resource
- Dialog box
- Multiple Document Interface - MDI

Menu and Resource

Get Started

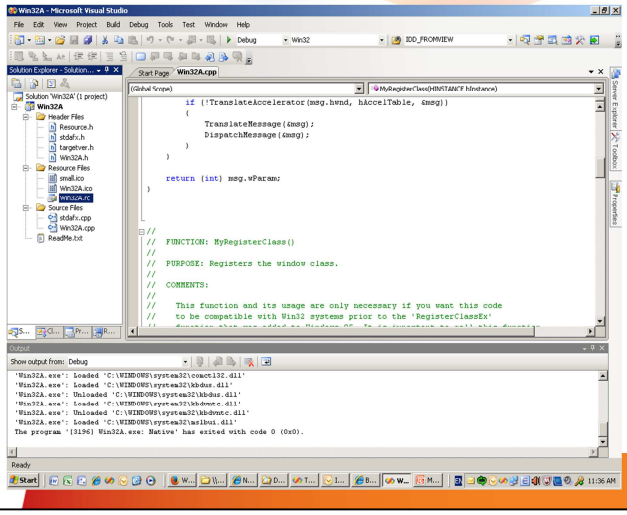
- Icons, cursors, menus, dialog boxes are all related. They are all type of Windows "**resources**"
- Resources are data and they are often stored in a program's .EXE file, but they do not reside in the executable program's data area
- Windows provides functions that explicitly or implicitly load a program's resources into memory so that they can be used
- List of resources's style :
  - Icons
  - Cursors
  - Text Strings
  - Custom resources
  - Menus
  - Keyboard accelerators
  - Dialog boxes
  - Bitmaps

Icons, Cursors, Strings, and Custom Resources

- One of benefits of using resources is that many components of a program can be bound into the program's .EXE file
- Without the concept of resource, a binary file such as an icon image would probably have to reside in a separate file that the .EXE would read into memory for using. Or the icon would have to defined in the program as an array of bytes
- As an resource, the icon is stored in a separate editable file but is bound into the .EXE file during the build process

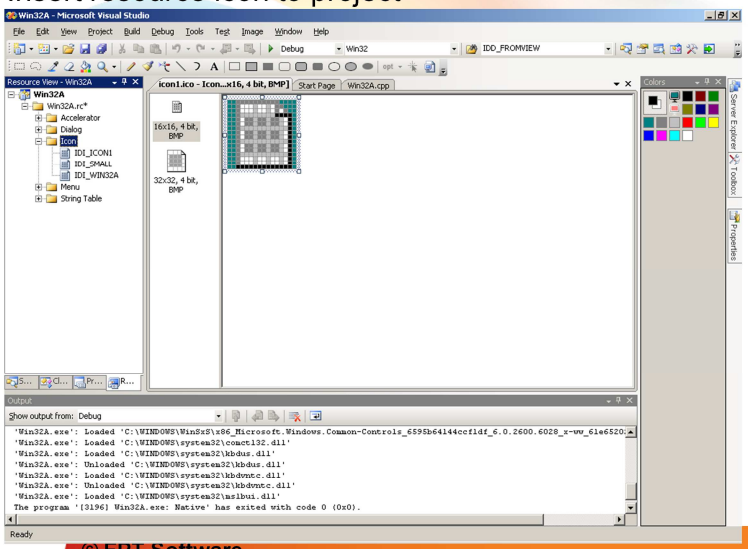
Create icon in a program

- 1. Create an Win32 Application project name Win32A
- 2. Open resource in visual studio environment



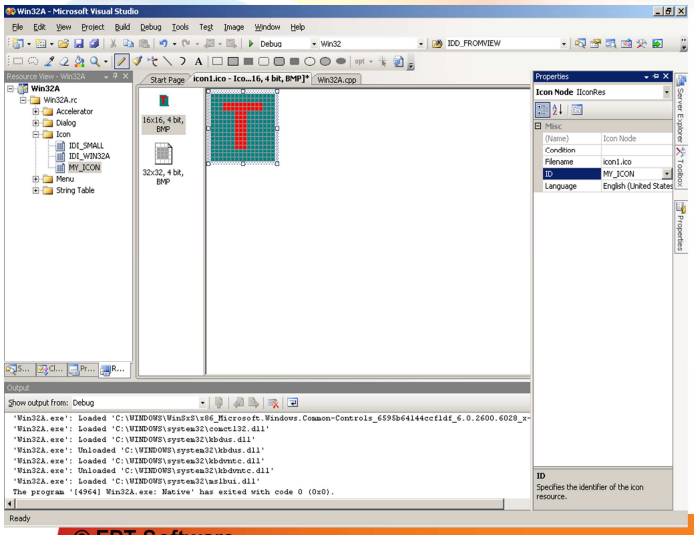
Create icon in a program

- 3. Insert resource icon to project



Create icon in a program

- 4. Edit Icon like below and change ID to MY_ICON in Properties



Create icon in a program

- Step 5 : Write code for load icon
  - Load icon by calling to LoadIcon function :

```
HICON LoadIcon(
    HINSTANCE   hInstance,      // handle to application instance
    LPCTSTR     lpIconName      // name string or resource identifier
    );
```

- If define icon identifier by identifier number :

```
HICON hIcon = LoadIcon (hInstance, MAKEINTRESOURCE
        (MY_ICON));
```

Create icon in a program

- **RESOURCE.H (excerpts) :**

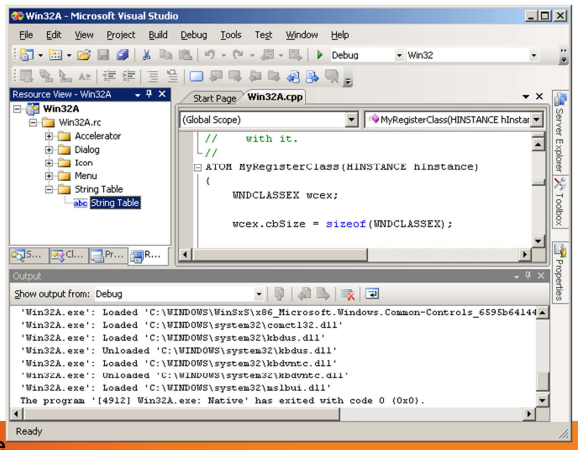
```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by IconDemo.rc
//
#define MY_ICON                 129
```

- **ICONDEMO.RC (excerpts) :**

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#define APSTUDIO_READONLY_SYMBOLS
// Generated from the TEXTINCLUDE 2 resource.
#include "afxres.h"
// Icon
// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
MY_ICON             ICON                    "icon1.ico"
```

String Table

- The character string resources (String table) are primarily for easing the translation of your program to other languages
- If the text in resource script file is translated to another language, all you need to do to create a foreign-language version of your program is re-link the program



String table in resource file

- In resource script file (.rc)

```
STRINGTABLE
BEGIN
    IDS_APP_TITLE       "Win32A"
    IDC_WIN32A          "WIN32A"
END
```

- Use LoadString function for copy a string resource into a buffer in the program's data segment :

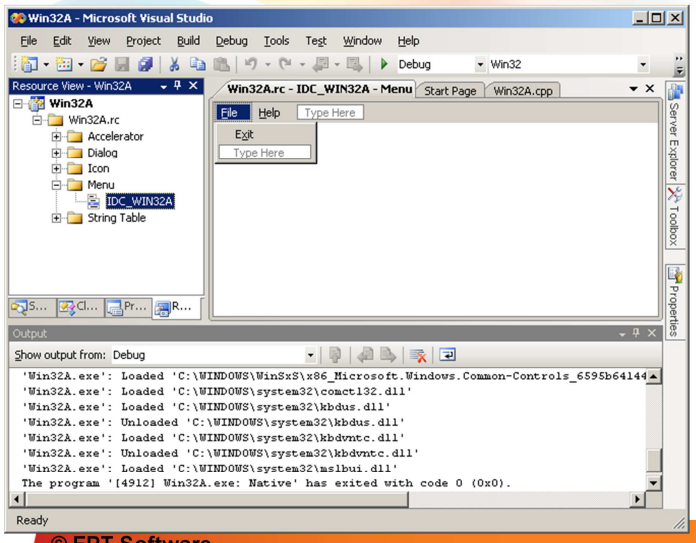
LoadString (HINSTANCE, UINT, LPTSTR, int);

Menus

- There are two main ways to create menu:
  - ✓ Create menu in design time
  - ✓ Create menu in run time (programmatically)

Create menu in design

- Double click on menu



Create menu programmatically

- Example Code :

```
hMenu = CreateMenu () ;

hMenuPopup = CreateMenu () ;

AppendMenu (hMenuPopup, MF_STRING,    IDM_FILE_NEW,     "&New") ;
AppendMenu (hMenuPopup, MF_STRING,    IDM_FILE_OPEN,    "&Open...") ;
AppendMenu (hMenuPopup, MF_STRING,    IDM_FILE_SAVE,    "&Save") ;
AppendMenu (hMenuPopup, MF_STRING,    IDM_FILE_SAVE_AS, "Save
&As...") ;
AppendMenu (hMenuPopup, MF_SEPARATOR, 0,                NULL) ;
AppendMenu (hMenuPopup, MF_STRING,    IDM_APP_EXIT,     "E&xit") ;

AppendMenu (hMenu, MF_POPUP, hMenuPopup, "&File") ;
```

Other Menu Commands

- Redrawing menu by calls to DrawMenuBar function :

  DrawMenuBar (HWND); // Handle of window

- Get handle of one popup menu by calls to GetSubMenu function :

  HMENU hMenuPopup = GetSubMenu (hMenu, iPosition);

- Get ID of one menu item in a popup menu by calls to GetMenuItemID function :

  Id = GetMenuItemID (hMenuPopup, iPosition);

- Get current flags of one menu item :

  iFlags = GetMenuState (hMenu, id, iFlag);

- Destroy a menu :

  DestroyMenu (hMenu);

Dialog box

Fundamental Concepts

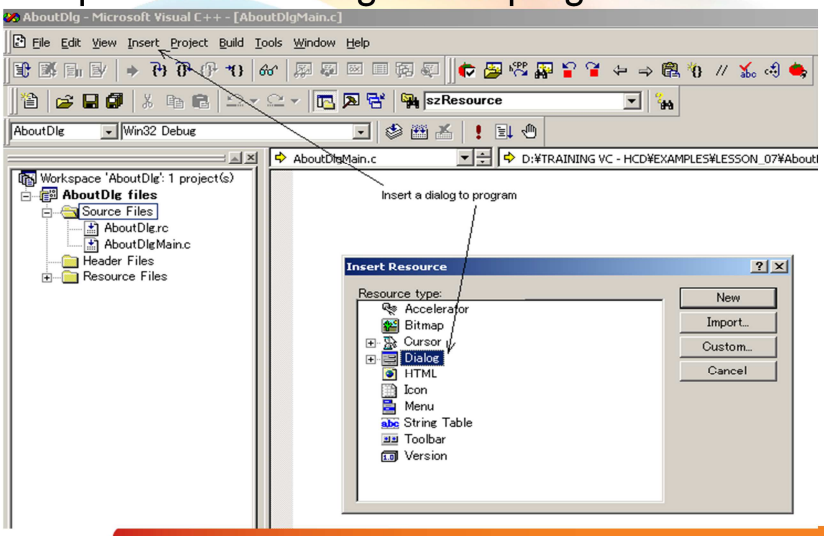
- Dialog boxes are windows that are most often used for obtaining additional input from the user beyond what can be easily managed through a menu
- Dialog box contains some controls elements such as: button, list box, combo box, edit box, …
- Dialog box is one type of resource
- Dialog boxes also have procedures that handling messages sent to them. These procedures are called dialog procedures.

Modal Dialog Boxes

- When a modal dialog box is displayed, the user can not switch between the dialog box and another window in program
- However, the user can switch to another program while the dialog box is displayed
- Insert a dialog box to program is similar to inserting an icon, a menu or other resources
- Design dialog box by using dialog box design tools provided by Microsoft Visual C++

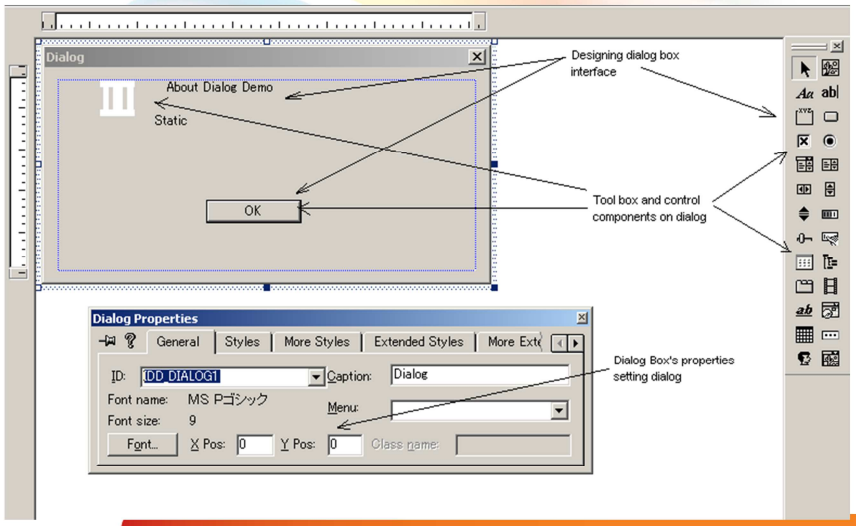
Create Dialog Boxes

- Step 1 : Insert dialog box to program



Create Dialog Boxes

- Step 2 : Design Dialog Box Interface



The Dialog Box Procedure

- The dialog box procedure within program handles messages sent to the dialog box. Although the dialog box procedure looks like a window procedure, it is not a true window procedure
- Example of dialog box procedure :

```
BOOL CALLBACK AboutDlgProc (HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {
    switch (message)   {
        case WM_INITDIALOG :
            return TRUE ;
        case WM_COMMAND :
          switch (LOWORD (wParam))  {
                                    case IDOK :
                                    case IDCANCEL :
                                EndDialog (hDlg, 0) ;
                                    return TRUE ;
          }
          break ;
    }
     return FALSE ;
}
```

The Dialog Box Procedure

- Some different things between window procedure and dialog box procedure :
  - Window procedure returns an ***LRESULT***; a dialog procedure returns a ***BOOL*** value
  - Window procedure calls ***DefWindowProc*** function if it does not process a particular message; a dialog box procedure returns ***TRUE*** if it processes a message and ***FALSE*** if it does not
  - A dialog box procedure does not need to process ***WM_PAINT*** or ***WM_DESTROY*** messages ; it also will not receive ***WM_CREATE*** message, instead, the dialog box procedure perform initialization during the special ***WM_INITDIALOG*** message

The Dialog Box Procedure

- The ***WM_INITDIALOG*** is the first message the dialog box procedure receives. This message is sent only to dialog box procedure
- The ***WM_COMMAND*** messages are sent from child window controls on dialog box to dialog box procedure when user interacts with them
- The messages for a modal dialog box do not go through program's message queue

Invoking The Dialog Box

- Invoking the dialog box by a calls to ***DialogBox*** macro

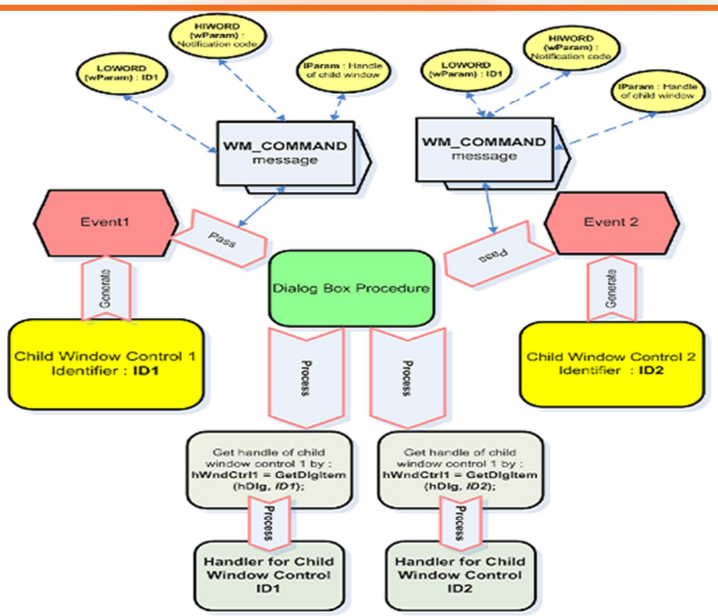
```
INT_PTR     DialogBox(
  HINSTANCE     hInstance,      // handle to module
  LPCTSTR       lpTemplate,     // dialog box template
  HWND          hWndParent,     // handle to owner window
  DLGPROC       lpDialogFunc    // dialog box procedure
);
```

- Even when the dialog box is displayed, ***WndProc*** can continue to receive messages by calling to the ***SendMesssage*** function in the dialog box procedure :

```
SendMessage (GetParent (hDlg), . . .);
```

- The ***DialogBox*** macro does not return control to Windows until the specified callback function terminates the modal dialog box by calling the ***EndDialog*** function

Working with Dialog Box Controls



The Modeless Dialog Boxes

- Modeless dialog boxes allow the user to switch between the dialog box and the window that created it as well as between the dialog box and other programs
- Modeless dialog boxes are created using ***CreateDialog*** :

```
HWND CreateDialog(
  HINSTANCE     hInstance ,              // handle to module
  LPCTSTR       lpTemplate ,     // dialog box template name
  HWND          hWndParent ,    // handle to owner window
  DLGPROC       lpDialogFunc     // dialog box procedure
);
```

- The difference is that the ***CreateDialog*** function returns immediately with the window handle of the dialog box
- Unlike messages to modal dialog boxes and message boxes, messages to modeless dialog boxes come through your program's message queue

Message Loop

- Altering the modeless dialog box's messages in message loop of window procedure

```
while (GetMessage (&msg, NULL, 0, 0)) {
    if (hDlgModeless == 0 ¦¦ !IsDialogMessage (hDlgModeless, &msg))    {
        TranslateMessage (&msg) ;
        DispatchMessage  (&msg) ;
    }
}
```

- If use keyboard accelerators :

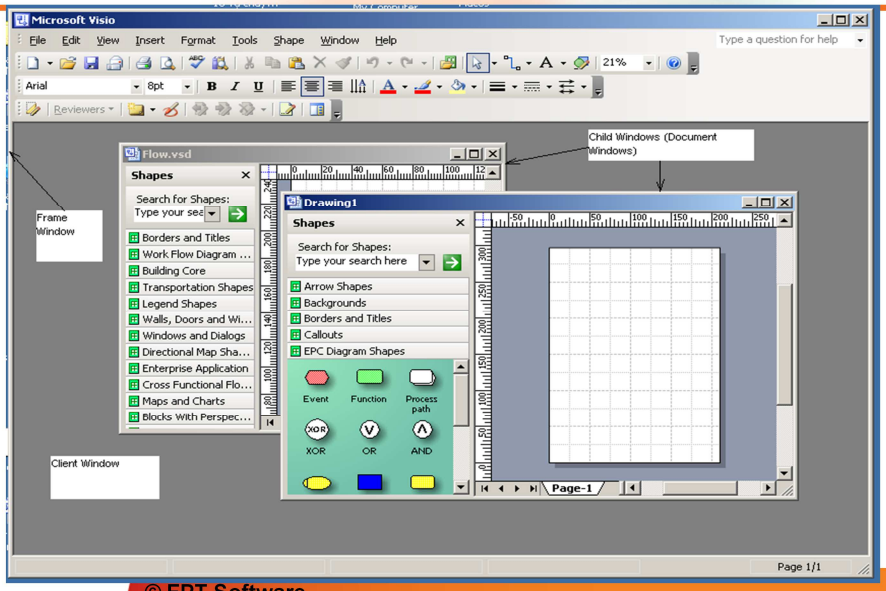
```
while (GetMessage (&msg, NULL, 0, 0)) {
    if (hDlgModeless == 0 ¦¦ !IsDialogMessage (hDlgModeless, &msg)) {
        if (!TranslateAccelerator (hwnd, hAccel, &msg))  {
            TranslateMessage (&msg) ;
            DispatchMessage  (&msg) ;
        }
    }
}
```

Multiple Document Interface

MDI – Fundamental Concepts

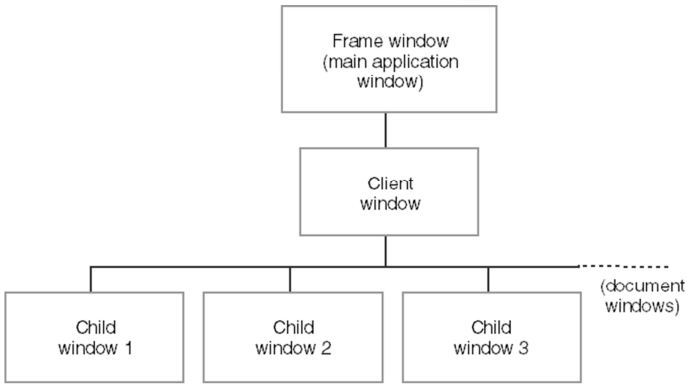
- The Multiple-Document Interface (MDI) is a specification for applications that handle documents in Microsoft Windows
- The Multiple-Document Interface (MDI) describes a window structure and user interface that allow the user to work with multiple documents within a single application

MDI - Elements



MDI - Elements

- The parent-child hierarchy of a Windows MDI application



Create MDI Application

- Declare frame window procedure, child windows procedures

  LRESULT CALLBACK ***FrameWndProc*** **(**HWND**,** UINT**,**
  WPARAM**,**
  LPARAM);

  LRESULT CALLBACK ***ChildWnd1Proc*** **(**HWND, UINT,
  WPARAM,
  LPARAM);

  LRESUTL CALLBACK ***ChildWnd2Proc*** (HWND, UINT,
  WPARAM,
  LPARAM);
  . . .
- Register frame window, client window, child windows classes

Create MDI Application

- Load menu for frame window from resource script
- Load keyboard accelerator table (if defined)
- Create frame window by calls ***CreateWindow*** function
- Get client window handle by calls ***GetWindow*** function

  ***hwndClient* = GetWindow (*hwndFrame*, *GW_CHILD*) ;**

- Change message loop

```
while (GetMessage (&msg, NULL, 0, 0)) {
  if (!TranslateMDISysAccel (hwndClient, &msg) &&
      !TranslateAccelerator (hwndFrame, hAccel, &msg)) {
          TranslateMessage (&msg) ;
          DispatchMessage (&msg) ;
  }
}
```

Frame Window Procedure

- Declare a ***CLIENTCREATESTRUCT*** variable :

```
CLIENTCREATESTRUCT clientCreate;
// Definition of CLIENTCREATESTRUCT :
typedef struct tagCLIENTCREATESTRUCT {
    HANDLE      hWindowMenu;
    UINT        idFirstChild;
}   CLIENTCREATESTRUCT, *LPCLIENTCREATESTRUCT;
```

- Declare a ***MDICREATESTRUCT*** variable :

```
MDICREATESTRUCT mdiCreate;
// Definitiion of MDICREATESTRUCT
typedef struct tagMDICREATESTRUCT {
   LPCTSTR      szClass;
   LPCTSTR      szTitle;
   HANDLE       hOwner;
   int          x;
   int           y;
   int          cx;
   int          cy;
   DWORD        style;
   LPARAM       lParam;
} MDICREATESTRUCT, *LPMDICREATESTRUCT;
```

Frame Window Procedure

- Handle the ***WM_CREATE*** message :
  - Setting data for ***clientCreate*** struct
  - Create client window by calls to ***CreateWindow***

```
hwndClient = CreateWindow (TEXT ("MDICLIENT"),
NULL,WS_CHILD | WS_CLIPCHILDREN | WS_VISIBLE, 0, 0,
0, 0, hwnd, (HMENU) 1, hInst, (PSTR) &clientcreate) ;
```

- Handle the ***WM_COMMAND*** message :
  - Setting data for ***mdiCreate*** struct
  - Create child window control by calls to ***SendMessage***

```
hwndChild = (HWND) SendMessage (hwndClient,
                    WM_MDICREATE, 0, (LPARAM )
                                (LPMDICREATESTRUCT)
&mdicreate) ;
```

Frame Window Procedure

- Retrieve the handle of the active MDI child window :

```
hWndChild = (HWND) SendMessage (hWndClient,
                                WM_MDIGETACTIVE, 0, 0 );
```

- Close the MDI child window :

```
SendMessage (hWndChild, WM_QUERYENDSESSION, 0,
             0 );

SendMessage (hWndClient, WM_MDIDESTROY,
(WPARAM) hWndChild, 0 );
```

- Pass the messages that no need to process in program to ***DefFrameProc***

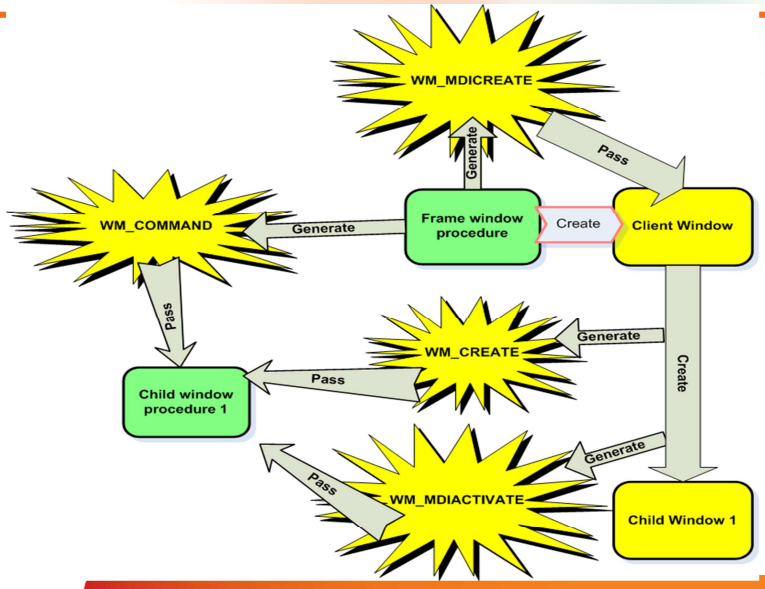
```
return DefFrameProc (hWnd, hWndClient, message,
                     wParam, lParam);
```

MDI Child Window Procedure

- MDI child window procedure is similar to other window procedures. It processes messages that are generated on child window or messages that are sent to child window from client window, main frame window
- With messages that are no need to process, pass to ***DefMDIChildProc***

return DefMDIChildProc (**hWnd**, **message**, **wParam**, **lParam**);

Frame – Client - Child




Q&A