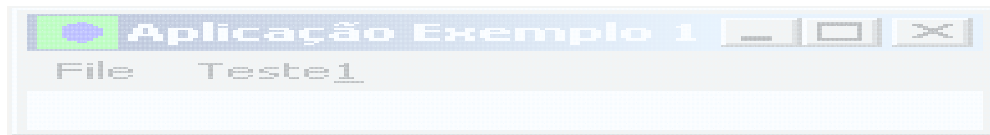
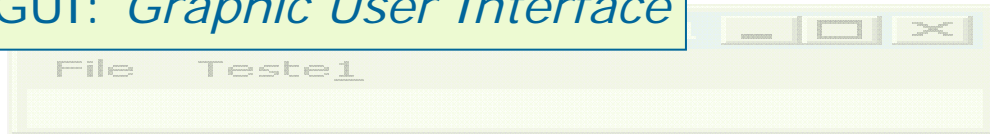


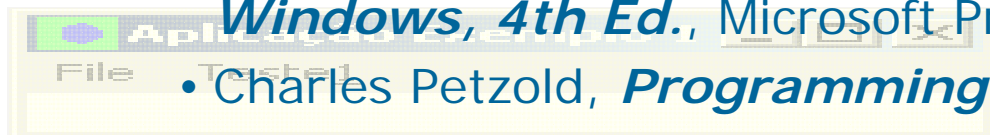
GUI: *Graphic User Interface*



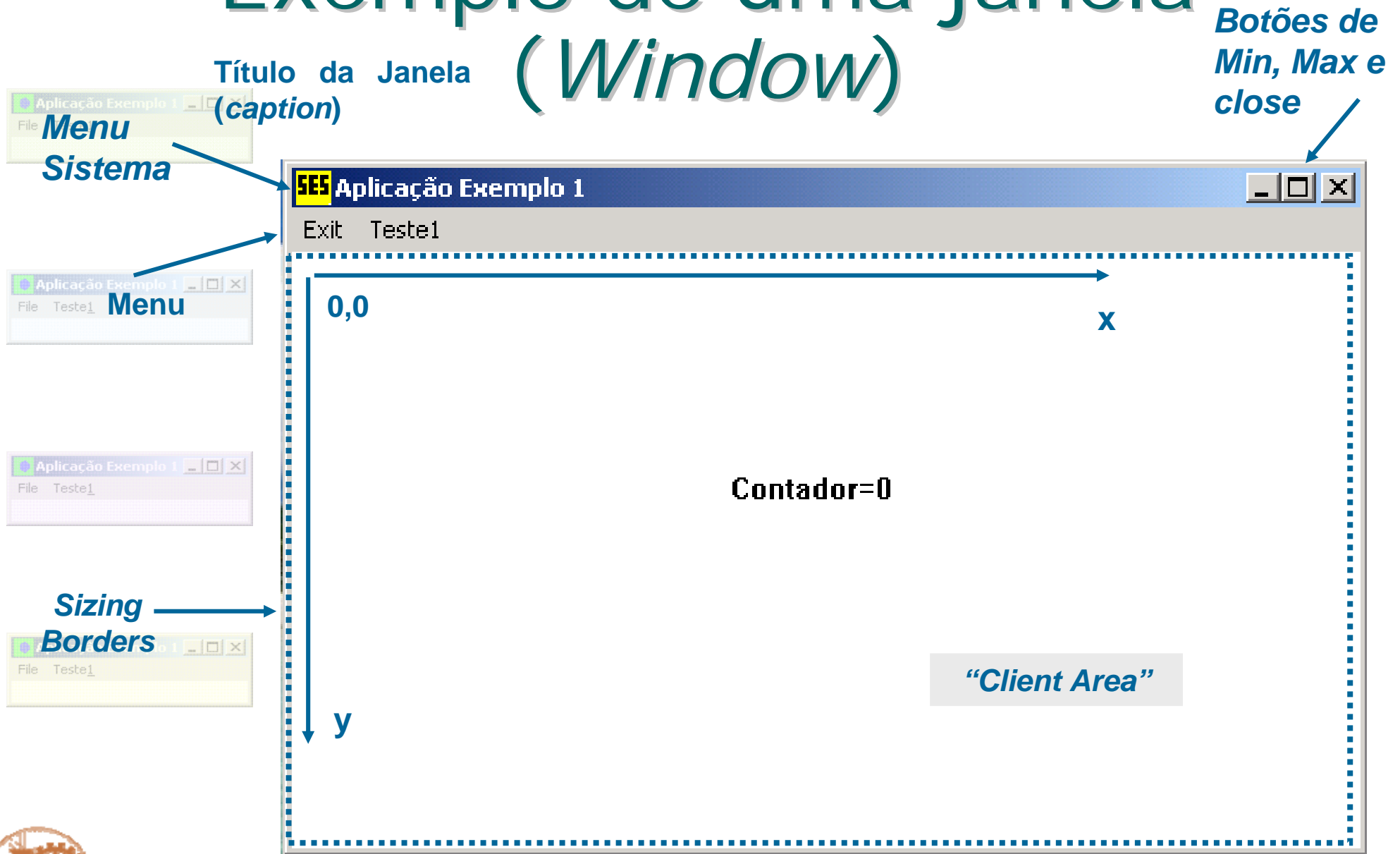
Modo GUI na API Win32



- Jeffrey Richter, *Programming Applications for Microsoft Windows, 4th Ed.*, Microsoft Press, 1999 [cap. 26]
- Charles Petzold, *Programming Windows, the definitive guide to the Windows 98 API*, Microsoft Press, 1998 [GUI API]
- MSDN: ms-help://MS.VSCC.v80/MS.MSDN.v80/MS.WIN32COM.v10.en/winui/winui/windowsuserinterface/windowui.htm



Exemplo de uma janela (*Window*)

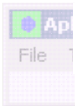


Entry point de um programa baseado em janelas (*Windows-based program*)



WinMain

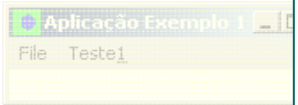
```
int APIENTRY _tWinMain(
    HINSTANCE hInstance, // handle de id desta instância do programa
    HINSTANCE hPrevInstance, // 0, existe para compatibilidade
    LPSTR lpCmdLine, // linha de comandos
    int nCmdShow) // como a janela deve ser mostrada inicialmente
```



Versão ASCII **WinMain** - versão UNICODE **wWinMain**

Valores para o campo **nCmdShow**

SW_SHOWNORMAL - Mostra a Janela normalmente
SW_SHOWMAXIMIZED - Mostra a Janela Maximizada
SW_SHOWMINIMIZED - Mostra a Janela Minimizada
SW_SHOWMINNOACTIVE - Mostra a Janela Minimizada e não activa
Outros valores possíveis: SW_HIDE, SW_RESTORE, SW_SHOW, SW_SHOWNA, SW_SHOWNOACTIVATE



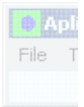
Janela Activa – é a janela que tem o *focus*



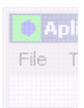
Windows-based programs



- Uma *Window* é uma área rectangular no écran que recebe *input* do utilizador e que mostra *outputs* sob a forma textual ou gráfica.



- O *input* é recebido sob a forma de mensagens
 - As acções do utilizador sobre o teclado e do *mouse* originam mensagens que representam a ocorrência desses eventos



- As mensagens são depositadas em "*messages queues*"
 - Que estão associada à janela onde o evento ocorreu



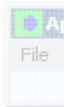
- As *threads* retiram as mensagens das *messages queues* e processam-nas
 - De forma a estruturar o processamento das mensagens, este processamento é realizado por uma função de "*dispatch*"



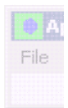
Windows, threads e message queues



- Cada *thread* tem uma *message queue*
 - Quando uma *thread* chama uma função relacionada com a Interface Gráfica (como `CreateWindow`) o sistema cria-lhe uma "*message queue*"



- A *thread* quando cria uma *Window* fica seu "*Owner*",
 - a *Window* existe enquanto a *thread* existir



- Os eventos associados à *Window*, vindos por exemplo do Rato ou do Teclado, são colocados na *message queue* da *thread owner* da *Window*



- Uma *Window* fica associada:
 - à queue da *thread* que a criou – onde são depositadas as msgs
 - a uma função "*WndProc*" (função de dispatcher)



Estrutura de um aplicação com janela gráfica



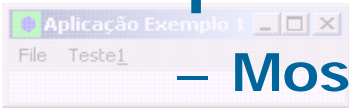
- Criar uma janela

- Registrar uma Classe para a Janela - RegisterClasseEx(...)



- Criar a Janela – CreateWindow(...)

- Apresentar a janela



- Mostrar a Janela – ShowWindow(...)

- Actualizar a área da janela - UpdateWindow

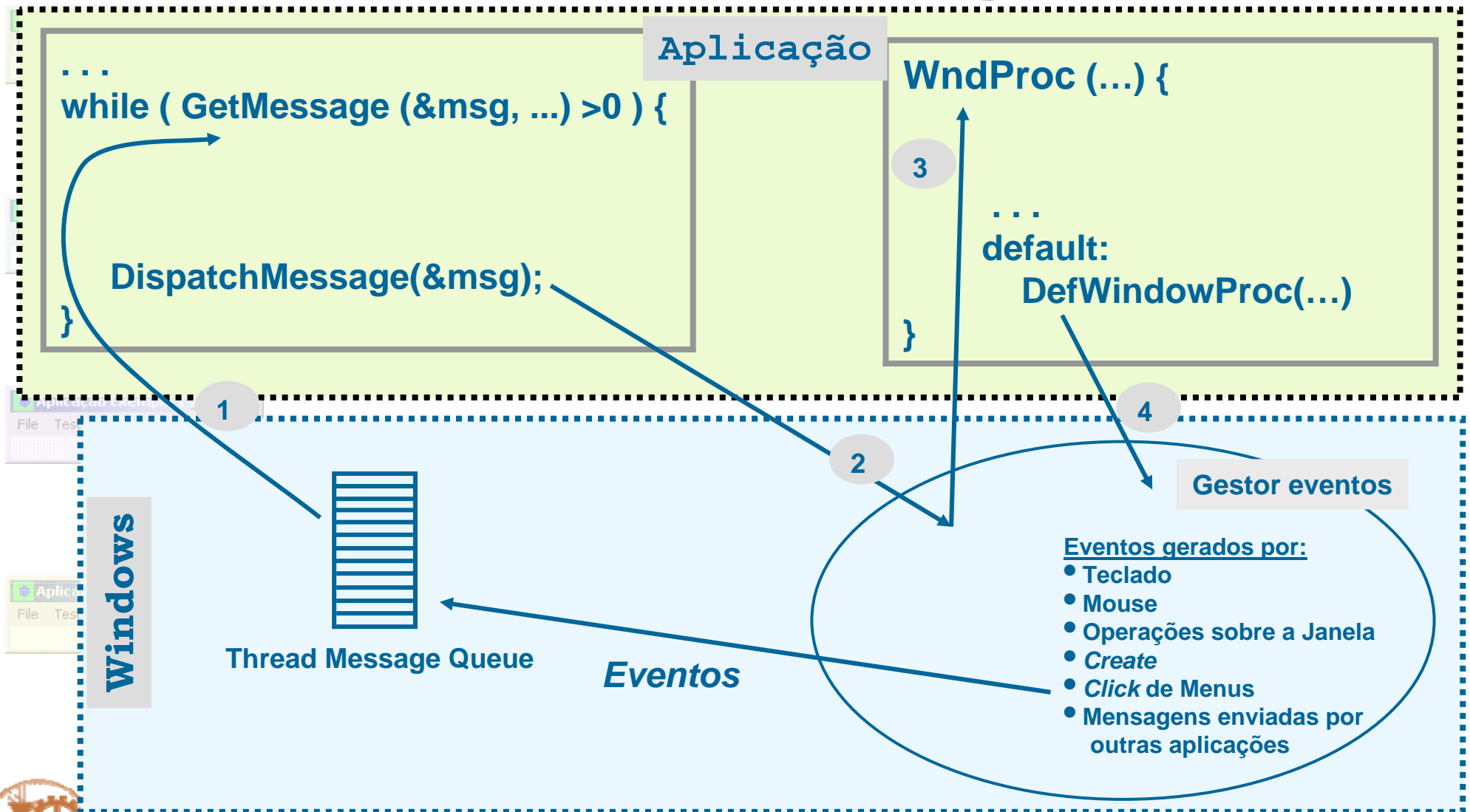


- *Message Loop*

- Terminar aplicação



Tratamento de Eventos numa Aplicação (*message loop*)



MyFirstWindow app

Main e Message Loop

```
#include <windows.h>

int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow) {
    HWND hwnd = MyCreateWindow(TEXT("The title of my window"),
                               WndProc, hInstance, 240, 120);

    ShowWindow(hwnd, nCmdShow); // para mostrar a window
    UpdateWindow(hwnd);         // para escrever a "Client area" da window

    MSG msg;
    while(GetMessage(&Msg, NULL, 0, 0) > 0) {
        DispatchMessage(&msg);
    }
    return (int)Msg.wParam;
}
```

DispatchFunction - WinProc

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam){
    switch(msg) {
        case WM_LBUTTONDOWN:
            MessageBox(NULL, TEXT("WinProc"), TEXT("Click"), MB_ICONEXCLAMATION|MB_OK);
            return 0;
        default: return DefWindowProc(hwnd, msg, wParam, lParam);
    }
}
```


Mensagens e MSG struct



Uma mensagem é uma instância da estrutura **MSG**



MSG struct

```
typedef struct {  
    HWND    hwnd;        // handle da window onde ocorreu o evento  
    UINT    message;      // tag identificadora da mensagem  
    WPARAM wParam;        // parâmetro  
    LPARAM lParam;        // parâmetro  
    DWORD   time;         // "time" na altura em que a msg foi colocada  
    POINT   pt;           // coordenadas do mouse, quando ocorreu a msg  
} MSG, *PMSG;
```

ID's de mensagens, exemplos:

WM_CREATE, WM_PAINT, WM_DESTROY,
WM_MOUSEMOVE, WM_LBUTTONDOWN, WM_CHAR



Queued and non-queued messages



- *Queued messages (message loop messages)*

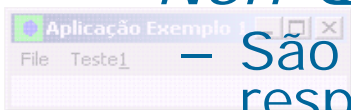
- São mensagens que são colocadas, pelo sistema, na *queue* de mensagens, e que são retiradas pelo GetMessage



- “*Posted*” to a *message queue*

- Exs: *Keystrokes, characters, mouse movements, mouse button clicks, timer, repaint and quit messages*

- *Non-Queued messages*



- São mensagens que o sistema entrega directamente à respectiva WinProc (*call*),

- “*sent*” *directly to the window procedure*

- Geralmente estas mensagens são resultantes da invocação de uma função do sistema relacionada com a *window*



- Exs:

- CreateWindow – WM_CREATE
- ShowWindow – WM_SIZE e WM_SHOWWINDOW
- UpdateWindow – WM_PAINT (esta mensagem também pode ser queued)
- DestroyWindow – WM_DESTROY



GetMessage

A função GetMessage retira mensagens da *message queue* da *thread*

GetMessage

```
BOOL GetMessage( // função bloqueante que retira uma msg
    LPMSG lpMsg, // ptr p. a struct que vai receber a msg
    HWND hWnd,   // hdl para a wind da qual vai ser lida a msg
    UINT wMsgFilterMin, // filtro inferior de mensagens
    UINT wMsgFilterMax ); // filtro superior de mensagens
```

-HWND a NULL: será retirada uma mensagem vinda de qualquer *Window* pertencente à *thread*, ou colocada por outra *thread*

- Filtragem de mensagens: `wMsgFilterMin = wMsgFilterMax = 0`, não há filtragem

Valor de retorno:

- >0, mensagem retirada com sucesso
- =0, Mensagem de QUIT retirada (WM_QUIT)
- 1, erro



Função WndProc



Função que efectua o processamento das mensagens



WndProc

```
LRESULT CALLBACK WndProc (  
    HWND    hWnd,    // Handle da window, que enviou a mensagem  
    UINT    uMsg,    // Tipo da mensagem  
    WPARAM  wParam,  // Primeiro Parâmetro da Mensagem  
    LPARAM  lParam   // Segundo Parâmetro da Mensagem  
)
```

Esta função é um CALLBACK, ou seja, não é uma função chamada, directamente, pela aplicação, mas sim chamada pelo sistema (quando ocorre um determinado evento)

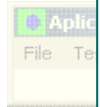
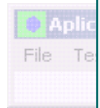
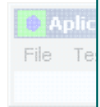


Esta função deve realizar o processamento das mensagens, mas também deve enviá-las para o *handler* do sistema de modo a que a funcionalidade normal da janela seja executada. Para tal deve chamar a função:

DefWindowProc(hWnd, uMsg, wParam, lParam));



Exemplo de uma WndProc



```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    HDC hdc; PAINTSTRUCT ps; RECT rect;

    switch(msg) {
        case WM_CREATE: . . . break;

        case WM_MOUSEMOVE: . . . break;

        case WM_PAINT:
            hdc = BeginPaint (hwnd, &ps);

            GetClientRect (hwnd, &rect);
            DrawText (hdc, TEXT("Hello, Windows XP!"), -1, &rect,
                DT_SINGLELINE | DT_CENTER | DT_VCENTER);
            EndPaint (hwnd, &ps);
            break;

        case WM_CLOSE: DestroyWindow(hwnd); break;

        case WM_DESTROY: PostQuitMessage(0); break;

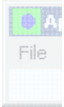
        default:
            return DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}
```



Descrição das Mensagens mais importantes



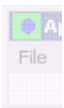
WM_COMMAND - enviada quando o utilizador selecciona um menu, quando um controle (CWC) envia uma notificação para a Janela principal ou quando uma Tecla Accelerator (*Alt X*) é premida



WM_CREATE - Enviada depois de uma janela ser criada

WM_DESTROY - Enviada depois de uma janela ser destruída

WM_CLOSE - Enviada quando uma janela ou aplicação deve terminar



WM_QUIT - significa que é requisitado à aplicação para terminar

WM_PAINT - enviada para que a *Client Area* da Janela seja redesenhada

WM_LBUTTONDOWN - Premido o botão esquerdo do rato



WM_LBUTTONUP - Largado o botão esquerdo do rato

WM_MOUSEMOVE - Movimento do rato

WM_KEYDOWN - foi premida uma tecla

WM_CHAR - Character recebido



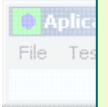
Translate message



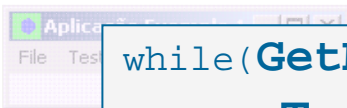
Para simplificar o tratamento dos eventos do teclado existe a função `TranslateMessage`.

Esta função transforma os eventos do teclado (que se referem ao premir e libertar uma tecla virtual) em caracteres.

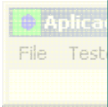
Colocando novas mensagens de **WM_CHAR** na *message queue*.



Message Loop com TranslateMessage



```
while(GetMessage(&Msg, NULL, 0, 0) > 0) {  
    TranslateMessage(&Msg);  
    DispatchMessage(&Msg);  
}
```



Mensagens retiradas da message queue

WM_KEYDOWN VK_ 'a'

TranslateMessage coloca na MQ: WM_CHAR 'a'

WM_KEYUP VK_ 'a'

WM_KEYDOWN VK_SHIFT

WM_KEYDOWN VK 'a'

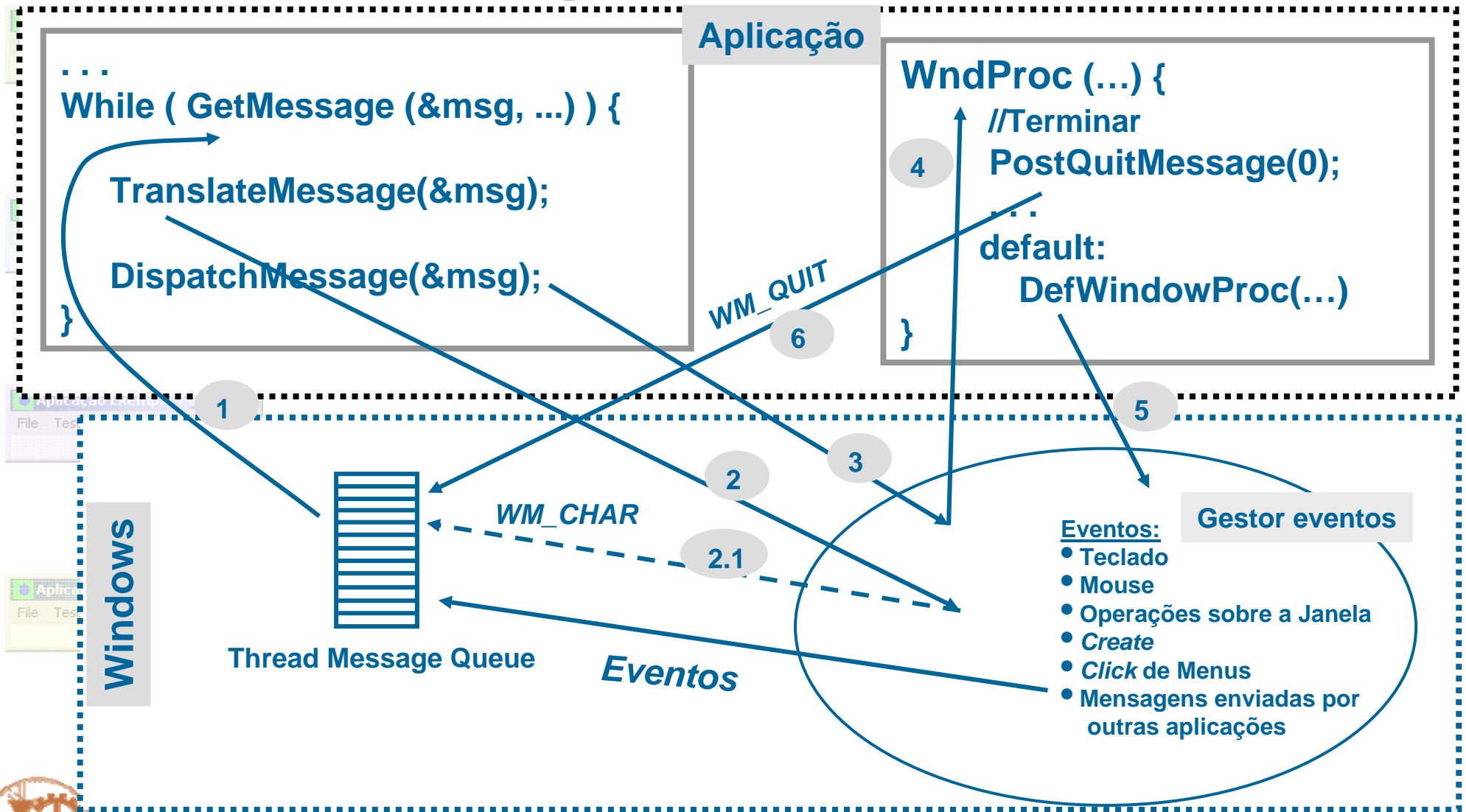
TranslateMessage coloca na MQ: WM_CHAR 'A'

WM_KEYUP VK 'a'

WM_KEYUP VK_SHIFT



Tratamento de Eventos (mensagens) numa Aplicação



Criar uma janela (3 passos)

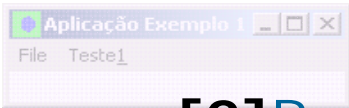


Na win32 a criação de uma janela tem os seguintes passos:

– **[1]** Criar um tipo de janela (**WNDCLASSEX**) – *Window class*



- Define as características principais
 - Cap. de redimensionamento, *icons*, cursor, cor de bg, menu
- É identificado por um nome
- É associado a uma **WinProc** (função para tratar os eventos)



– **[2]** Registrar o tipo de janela (**RegisterClasseEx**)

– **[3]** Criar uma janela com base num tipo registado (**CreateWindow**)



- Cria a janela
- Define mais atributos da janela
 - Estilo da janela, coordenadas e dimensão



[1] e [2] – WNDCLASSEX



[1] Estrutura que define uma *class* de janela

WNDCLASSEX

```
typedef struct _WNDCLASSEX {
    UINT        cbSize;           // size em bytes da estrutura, sizeof(WNDCLASSEX).
    UINT        style;           // Combinação de estilos de classes de Janelas
    WNDPROC      lpfnWndProc;     // Pointer para função da Windows Procedure
    int          cbClsExtra;      // N° de bytes extra a alocar a seguir à estrutura
    int          cbWndExtra;      // N° de bytes ... a seguir à instância da window
    HINSTANCE    hInstance;      // Hnd para a instância com a WinProc desta classe
    HICON        hIcon;          // Handle para um large Icon. NULL -> icon por omissão
    HCURSOR      hCursor;        // Hnd para um Cursor. NULL -> app. define cursor Ondemand
    HBRUSH       hbrBackground;  // Hnd para o BG brush, ou valor de uma cor + 1
    LPCTSTR      lpszMenuName;    // Pointer para string com o nome do resource Menu
    LPCTSTR      lpszClassName;  // Ptr para string com o nome da Window Classe
    HICON        hIconSm;        // Ptr para um small icon, NULL (icon por defeito)
} WNDCLASSEX, *pWNDCLASSEX;
```

[2] Função que regista uma classe de janela

RegisterClassEx

```
ATOM RegisterClassEx( CONST WNDCLASSEX *lpwccx );
```



[3] – CreateWindow



[3] Função que cria uma janela, com base numa classe de janela



CreateWindow

```
HWND CreateWindow (  
    LPCTSTR lpClassName, // Nome de uma classe registada  
    LPCTSTR lpWindowName, // Nome (Título) da Janela  
    DWORD dwStyle,        // Estilo da Janela  
    int x,                // Posição Horizontal da Janela. Canto sup. esquerdo  
    int y,                // Posição Vertical da Janela. Canto sup. esquerdo  
    int nWidth,           // Largura da Janela em pixels  
    int nHeight,          // Altura da Janela em pixels  
    HWND hWndParent,      // Handle para a Janela Parent  
    HMENU hMenu,          // Handle para Menu. NULL se for para usar o da classe  
    HINSTANCE hInstance,  // Handle para a instância da aplicação  
    LPVOID lpParam        // NULL ou Pointer para uma estrutura com dados  
                        // de iniciação da Janela CREATESTRUCT  
);
```



Exemplo: MyCreateWindow

```
HWND MyCreateWindow(TCHAR * szWindowName, WNDPROC WndProc, HINSTANCE hInstance,
                    int xSize, int ySize) {
    [1] const TCHAR g_szClassName[] = TEXT("myWindowClass");
    WNDCLASSEX wc; HWND hwnd; int cxScreen, cyScreen, xPos=..., yPos=...;

    //Creating the Window Class and registering it
    wc.cbSize      = sizeof(WNDCLASSEX); wc.style          = 0;
    wc.lpfnWndProc  = WndProc;            wc.cbClsExtra      = 0;
    wc.hInstance    = hInstance;          wc.cbWndExtra      = 0;
    wc.hIcon        = LoadIcon(NULL, IDI_APPLICATION); // 32*32 large icon
    wc.hCursor      = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
    wc.lpszMenuName  = NULL;
    wc.lpszClassName = g_szClassName;
    wc.hIconSm       = LoadIcon(NULL, IDI_APPLICATION); // 16*16 small icon

    [2] if(!RegisterClassEx(&wc)) { ... } // erro

    // Creating the Window
    [3] hwnd = CreateWindow(g_szClassName,
        TEXT("The title of my window"), WS_OVERLAPPEDWINDOW, xPos, yPos,
        xSize, ySize, NULL, NULL, hInstance, NULL);
    if(hwnd == NULL) // erro
    return hwnd;
};
```



Funções *Show* e *Update* Window



Função que “mostra” uma janela

ShowWindow

```
BOOL ShowWindow(  
    HWND hWnd,           // handle da window  
    int nCmdShow );      // indica como a janela deve ser mostrada  
                        // utilizar o parâmetro recebido no _tWinMain
```



Função que actualiza a “*client area*” da *Window*

UpdateWindow

```
BOOL UpdateWindow(  
    HWND hWnd );         // handle da window
```

Esta função envia uma mensagem de WM_PAINT para a janela (caso a região em *update* esteja inválida).

A mensagem é entregue directamente à WinProc.



Resumo de uma *App*

Main e Message Loop

```
#include <windows.h>

int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow) {

    MSG Msg;  WNDCLASSEX wc = ...; // 1º definir a Window Class

    RegisterClassEx(&wc); // 2º registrar a Window Class
    HWND hwnd = CreateWindow(...); // 3º Criar a janela

    ShowWindow(hwnd, nCmdShow); // 4º Colocar a janela visível
    UpdateWindow(hwnd); // 5º fazer update à parte "client area" da janela

    while(GetMessage(&Msg, NULL, 0, 0) > 0) { // 6º retirar mensagens da queue
        DispatchMessage(&Msg); // 7º enviar as mensagens para a WinProc
    }
    return (int)Msg.wParam;
}
```

DispatchFunction - WndProc

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam) { ...
}
```

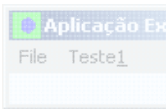


Como finalizar um programa



- *Click on Close button*

- O utilizador prime o botão para fechar a janela



- *O sistema coloca WM_CLOSE na msg queue*

- A aplic. ao receber WM_CLOSE executa o pedido de eliminar a janela

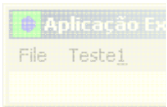
- *App. calls: DestroyWindow(hwnd)*



- *O sistema coloca WM_DESTROY na msg queue*

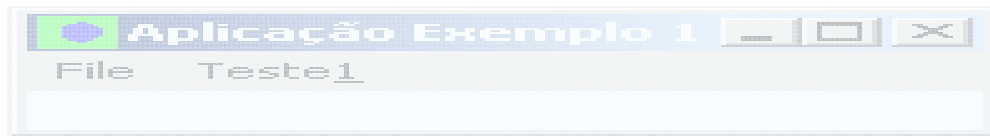
- Esta mensagem é colocada quando a janela já foi destruída
 - Caso a aplic. deseje terminar, deve dar indicação para o message loop terminar

- *App. calls: PostQuitMessage(0)*



- O GetMessage ao retirar a mensagem WM_QUIT, devolve 0
 - O message loop interpreta o 0 como sendo para terminar a aplicação

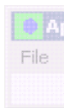
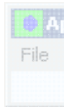




Painting and repainting



Painting and repainting



- A actualização da "*client area*" das janelas é da responsabilidade da aplicação
- A ocorrência de *creates*, *resizes*, *minimizes*, *maximizes*, etc., sobre uma janela, faz com que parte do seu conteúdo, (ou de outras janelas) tenha que ser (re)desenhado (*ainted*),
- Quando tal ocorre, o sistema coloca a área afectada como "inválida" e notifica a aplicação, enviando-lhe uma mensagem de WM_PAINT
- O programa após receber WM_PAINT, deve redesenhar a área inválida, e colocá-la válida
- O próprio programa também pode invalidar uma área a fim de forçar a sua actualização (InvalidateRect)



Conceito de Rectângulo Inválido



Invalidate/Validate Rect

InvalidateRect – esta função coloca um retângulo como inválido, e origina o “envio” da mensagem de WM_PAINT para a window

```
BOOL InvalidateRect(  
    HWND hWnd,           // handle to window  
    CONST RECT* lpRect,  // rectangle coordinates  
    BOOL bErase );       // erase background flag
```

ValidateRect – esta função coloca um retângulo como válido

```
BOOL ValidateRect(  
    HWND hWnd,           // handle to window  
    CONST RECT* lpRect,  // rectangle coordinates );
```

UpdateWindow – esta função chama directamente a WndProc associada à janela (com WM_PAINT), caso exista uma região inválida

```
BOOL UpdateWindow( HWND hWnd // handle to window );
```



Processamento do WM_PAINT



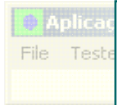
```
case WM_PAINT: // existe uma área inválida, que tem de ser desenhada e validada
    PAINTSTRUCT ps ;
    HDC hdc = BeginPaint (hwnd, &ps);
    TextOut(hdc, 20, 20, TEXT("OLA"), 3 );
    EndPaint (hwnd, &ps); // valida a região
    break;
```



```
HDC BeginPaint(                // devolve o Handle do Device Context
    HWND hwnd,                 // handle to window
    LPPAINTSTRUCT lpPaint ); // devolve paint information (PAINTSTRUCT)
```



```
BOOL EndPaint(
    HWND hwnd,                 // handle to window
    CONST PAINTSTRUCT *lpPaint ); // paint data
```



O **BeginPaint** prepara a janela para PAINT:

- Estabelece a área de clipping do DC para a zona inválida
- valida a zona inválida

- Se necessário apagar o background envia WM_ERASEBKGND para a janela e devolve uma PAINTSTRUCT preparada para auxiliar o PAINTing

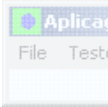
O **EndPaint** indica o fim do processamento do WM_PAINT.



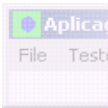
Estruturação de programas



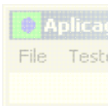
- Os programas devem estar estruturados de modo a que o processamento associado à mensagem WM_PAINT possa repor o conteúdo de qualquer parte inválida.



- Para tal é necessário que toda a informação necessária para redesenhar a client area esteja acessível ao processamento do WM_PAINT.




- Se após o processamento da mensagem WM_PAINT a região permanecer inválida e o *Windows* vai gerando sucessivamente mensagens de WM_PAINT.



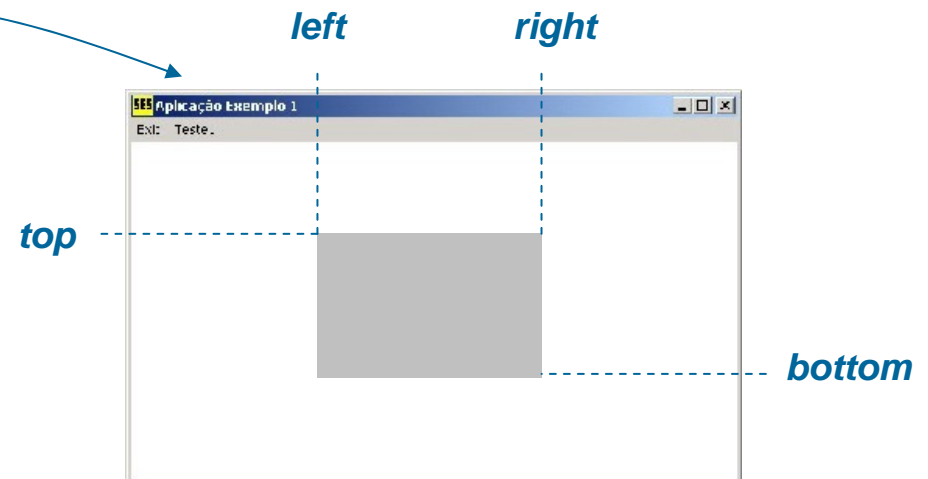
Estrutura PAINTSTRUCT

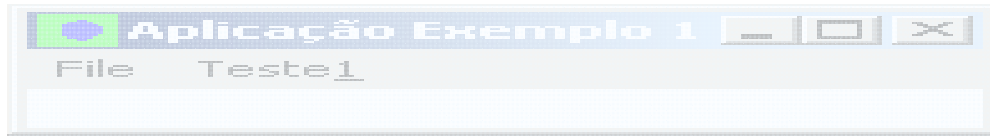


```
typedef struct tagPAINTSTRUCT {  
    HDC hdc;           // Handle Device Context usado para Paint  
    BOOL fErase;       // Indica se a aplicação deve apagar o background  
                        // Se a aplicação definiu um background brush, é devolvido FALSE  
    RECT rcPaint;     // Define o rectângulo da área que se encontra inválida  
    BOOL fRestore ;      // reservado  
    BOOL fIncUpdate ;    // reservado  
    BYTE rgbReserved[32] ; // reservado  
} PAINTSTRUCT ;
```

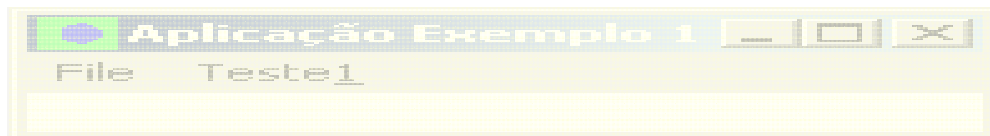


```
typedef struct _RECT {  
    LONG left;  
    LONG top;  
    LONG right;  
    LONG bottom;  
} RECT, *PRECT;
```

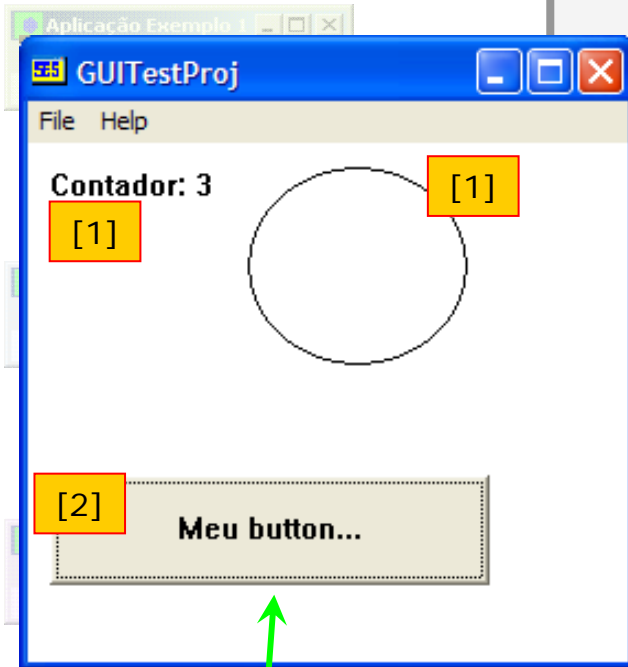




Exemplo de uma aplicação GUI



Exemplo



Aplicação Exemplo 1
File Teste1
CWC
(Child Window Control)
Janela Filha

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam){
    ...
    switch (message){
        case WM_CREATE:
            /* Inserir um Child Window Control [dremedios] */
            CreateWindow( TEXT("button"), TEXT("Meu button...")
                , WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 10, 150, 200, 50
                , hWnd, (HMENU) 1234 /* id do controlo */
                , ((LPCREATESTRUCT) lParam)->hInstance, NULL);

            buttonCounter = 0;
            break;
        case WM_COMMAND:
            switch (LOWORD(wParam)){
                ...
                /* tratamento do premir do botão 1234 */
                case 1234:
                    buttonCounter++;
                    /*forçar nova pintura da contagem*/
                    InvalidateRect(hWnd, &r, TRUE);
                    break;
                default:
                    return DefWindowProc(hWnd, message, wParam, lParam);
            }
            break;
        case WM_PAINT:
            hdc = BeginPaint(hWnd, &ps);
            /* Desenhar texto e figuras na client area */
            TCHAR textBuffer[80];
            _stprintf(textBuffer, TEXT("Contador: %d"), buttonCounter);
            DrawText(hdc, textBuffer, _tcslen(textBuffer), &r, NULL);
            Ellipse(hdc, 100, 10, 200, 100);
            EndPaint(hWnd, &ps);
            break;
        ...
    }
}
```



Como efectuar *input* e *output* na janela

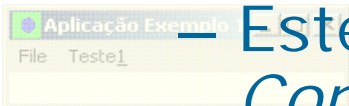


- [1] Desenhar linhas, figuras ou texto:
 - No tratamento do WM_PAINT desenhar com as primitivas: `LineTo(...)`, `Ellipse(...)`, `Rectangle(...)`, `DrawText(...)`, etc...

Ver MSDN



- [2] Desenhar controlos com comportamento já pré-definido:
 - Botões, campos de texto, etc...



- Estes controlos são janelas filhas (*Child Window Controls*) que se criam e colocam na *client area* da janela pai.

Ver Anexo 2



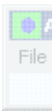
Edição de *Resources*

Janelas, Icons, *Dialog Boxes*, etc.



1) Edição em texto de *script* file (*.rc)

Exige muitos detalhes (coordenadas, dimensões etc.) sobre o *resource* (muito complexo)



2) Utilização do editor de *resources* do *Microsoft Developer Studio* (cria um *script*)

```
//Developer Studio resource script
```

```
...
```

```
// Menu
```

```
IDR_MENU1 MENU DISCARDABLE
```

```
BEGIN
```

```
    MENUITEM "E&xit",    ID_EXIT
```

```
    MENUITEM "Teste&1", ID_TESTE1
```

```
END
```

Script1.rc

```
// Icon
```

```
ID_ICON ICON DISCARDABLE "icon1.ico"
```

```
...
```

```
//Developer Studio include file
```

```
#define IDR_MENU1    101
```

```
#define ID_ICON      102
```

```
#define ID_EXIT      40001
```

```
#define ID_TESTE1    40002
```

```
...
```

resource.h



Tipos de Janelas de diálogo (Dialog Boxes - DB)

- Modal DB
- Modeless DB
- System Modal



Dialog Boxes

- Uma *dialog box* é uma janela (*window*)
- Geralmente são *popup windows* com *Child Window Controls (CWC)*
 - Com processamento de *input* incorporado dos CWC, com sequência de percurso por Tabs, com controle do focus, ...
 - Definidos numa "dialog box template" no "program's resource script file"
- Têm uma *dialog box procedure*
 - Iniciação das CWC
 - Processamento de mensagens originadas nos CWC
 - Terminação da *dialog box*
 - Tipicamente, não têm processamento de WM_PAINT, mouse e keyboard



Dialog Boxes

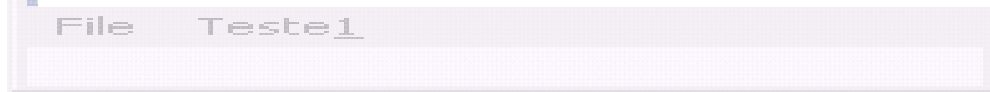


- Tipos de *Dialog boxes*:
 - **Modal** – o utilizador tem de encerrar a *dialog box*, antes de poder seleccionar outra janela da aplicação
 - **Modeless** – *Dialog Box* sem Parent Window, permite activar outras janelas da aplicação
 - **System Modal** – A *Dialog Box* permanece sempre como a janela principal (não permite mudar sequer para outra janela de outra aplicação)

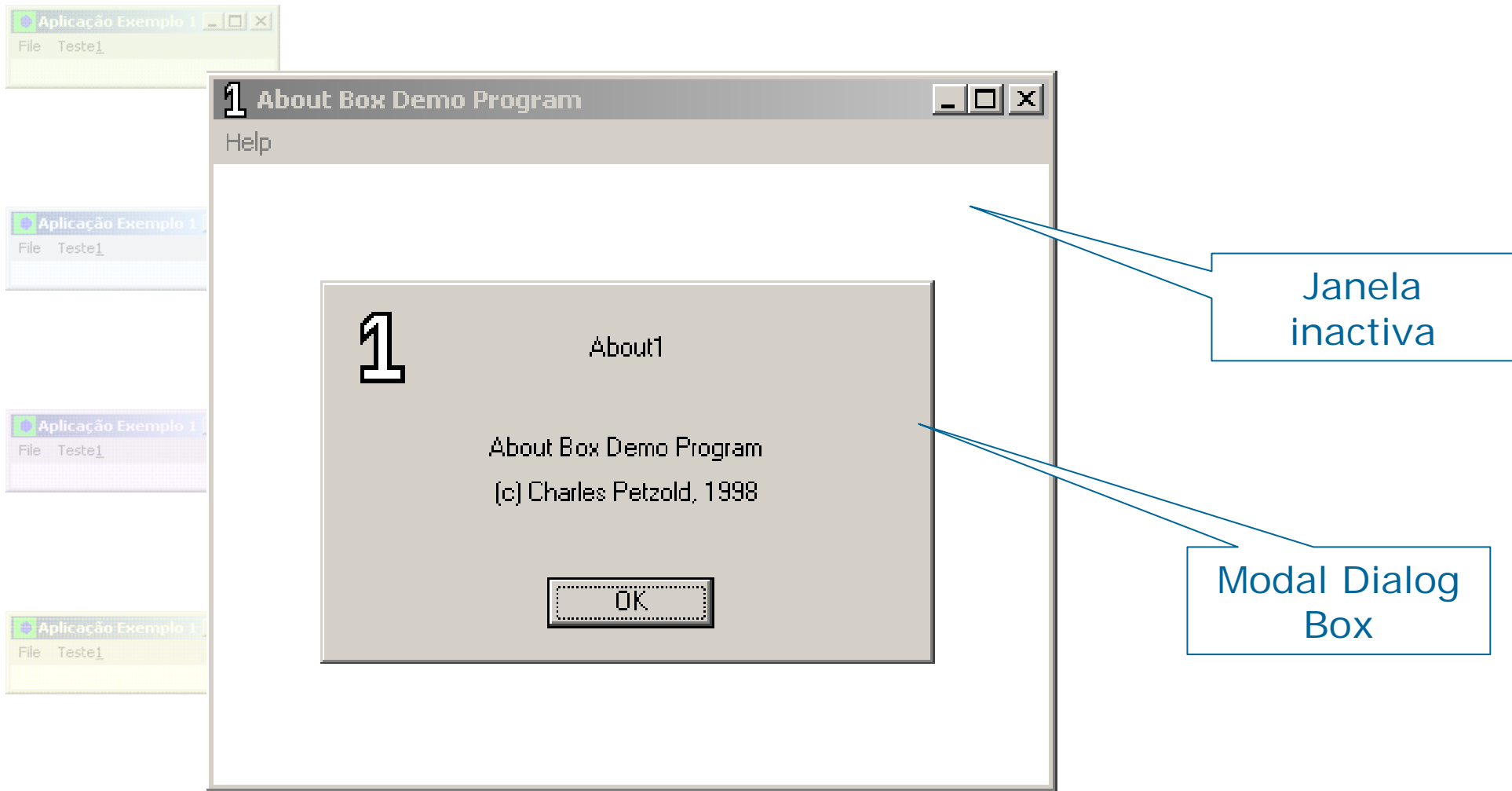


Tipos de Janelas de diálogo (Dialog Boxes - DB)

- **Modal DB**



Modal Dialog Box



Menu e DB *template* (VS - edição de *resources*)

Menu e Icon

```
ABOUT1 MENU
BEGIN
    POPUP "&Help"
    BEGIN
        MENUITEM "&About About1...", IDM_APP_ABOUT
    END
END

ABOUT1 ICON "About1.ico"
```

Dialog box template

```
ABOUTBOX DIALOGEX 32, 32, 180, 102
STYLE DS_SETFONT | DS_MODALFRAME | WS_POPUP
FONT 8, "MS Sans Serif", 0, 0, 0x0
BEGIN
    DEFPUSHBUTTON    "OK",IDOK,66,80,50,14
    ICON              "ABOUT1",IDC_STATIC,6,7,21,20
    CTEXT              "About1",IDC_STATIC,40,12,100,8
    CTEXT              "About Box Demo Program",IDC_STATIC,6,40,167,8
    CTEXT              "(c) Charles Petzold, 1998",IDC_STATIC,6,52,167,8
END
```

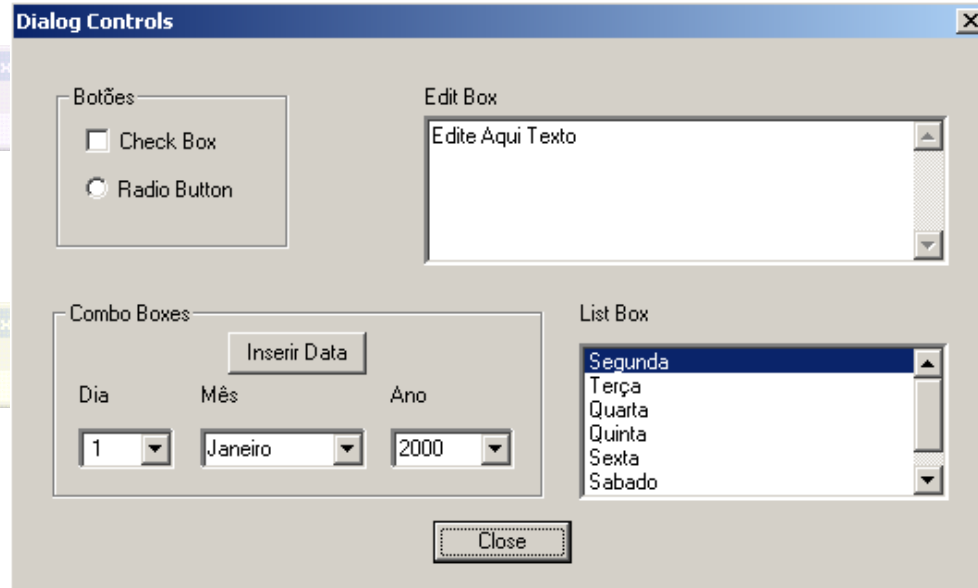
Nome da dialog box



DB Resource Controls

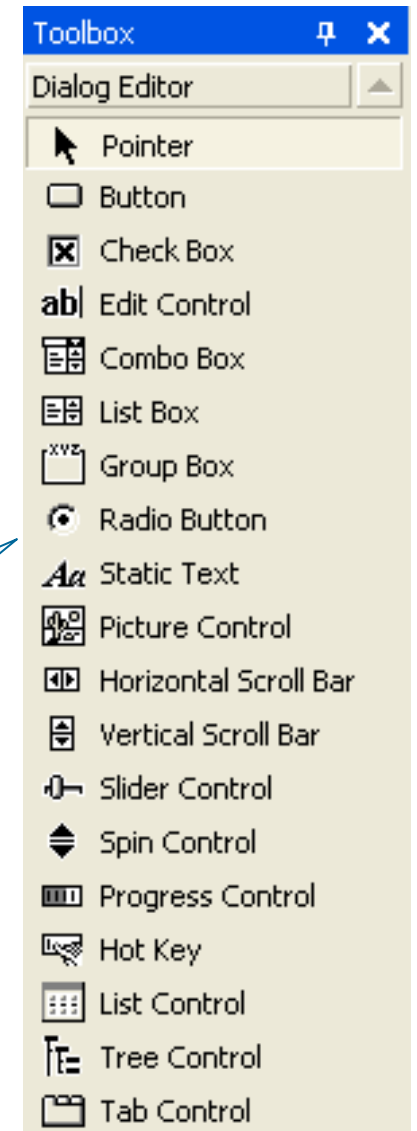
Controls: os componentes de uma Janela de diálogo

- Exemplos: button, edit, combo box, list box, group box, static text
- São objectos gráficos geralmente compostos por janelas pré-definidas e com uma funcionalidade de alto nível



Controls
disponíveis no
VS

Controls
inseridos numa
janela de
diálogo



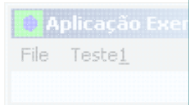
Criar e *Dialog Box Procedure*



Wnd Procedure "Main window"

```
case WM_COMMAND :  
    switch (LOWORD (wParam)) {  
        case IDM_APP_ABOUT :  
            DialogBox(hInstance, TEXT ("AboutBox"), hwnd, AboutDlgProc) ;  
            break ;
```

Nome da *dialog box*



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 ;  
}
```



Criar uma *Dialog Box*

DialogBox

```
int DialogBox(           // macro que cria uma Dialog Box a partir de um resource template
    HINSTANCE hInstance, // Handle do programa que contém o resource template
    LPCTSTR lpTemplate,   // String ou ID do template. MAKEINTRESOURCE do ID do resource
    HWND hWndParent,      // Handle da Janela Parent.
                          // NULL se a Dialog for a Janela Principal, Dialog Box MODELESS
    DLGPROC lpDialogFunc // dialog box procedure
);
```

- DialogBox não retorna o controlo até à Dialog Procedure (lpDialogFunc) invocar EndDialog
- Comportamento:
 - Chama CreateWindowEx(..) para criar uma *Window* do tipo *Dialog Box*.
 - Envia a mensagem WM_INITDIALOG para a função *Dialog Procedure*;
 - Desactiva a janela *parent* (MODAL *Dialog Box*) e inicia o *Message Loop*;
 - Quando a *Dialog Procedure* chama EndDialog, a Dialog Box é destruída,
 - termina o *Message Loop* e é activada a janela *parent* (se a *Dialog Box* é MODAL).
 - Retorna o valor nResult

Retorna:

nResult, retornado pela função EndDialog() quando termina a *Dialog Box*
0, erro o parâmetro hWndParent é inválido
-1, outro erro (chamar GetLastError)

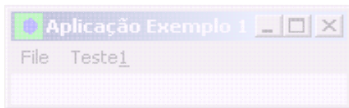


Dialog Box Procedure



Dialog Box Procedure

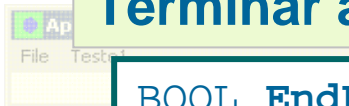
```
INT_PTR CALLBACK DialogProc (  
    HWND hwndDlg,    // handle to dialog box  
    UINT uMsg,        // mensagem  
    WPARAM wParam,    // Primeiro Parâmetro da mensagem  
    LPARAM lParam     // Segundo Parâmetro da mensagem  
);
```



Deverá retornar :

TRUE – mensagem processada

FALSE – mensagem não processada, indica para ser executada a acção por omissão associada à mensagem

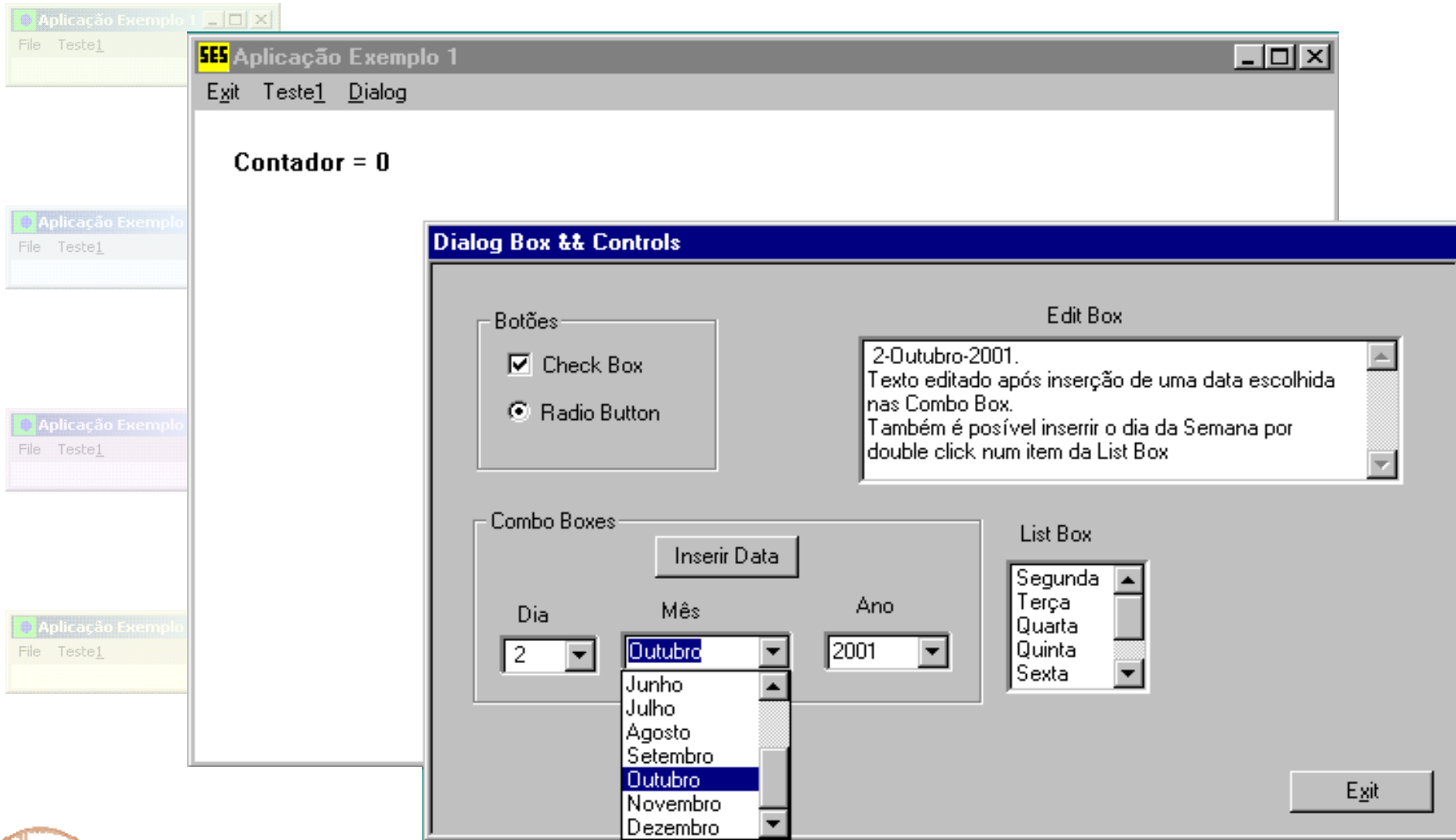


Terminar a Dialog Box

```
BOOL EndDialog(  
    HWND hDlg,        // Handle da Dialog Box a Terminar  
    INT_PTR nResult    // valor a retornar à função que criou a Dialog Box  
);
```



Exemplo de uma *Modal Dialog Box*



Criação da *Dialog Box* na Windows Procedure



```
LRESULT CALLBACK WndProc ( . . . ) {
    switch (uMsg) {
        case WM_COMMAND :
            switch ( LOWORD( wParam ) ) {
                ...
                case ID_DIALOG : // opção do menu que cria a dialog box
                    if (!DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG), hWnd, (DLGPROC)Dialog) )
                        MessageBox(hWnd,"Erro ao criar a Dialog Box","Debug",MB_OK);
                    break;
                ...
            } // switch ( LOWORD( wParam ) )
            break;
        ...
    } // switch uMsg

    return msg.wParam;
}
```

GetDlgItem

```
HWND GetDlgItem( HWND hDlg, int nIDDlgItem ); // devolve o handle do item
```



Dialog Procedure

Interacção com os CWC

```
LRESULT CALLBACK Dialog(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {  
    // Data actual  
    TCHAR diaActual[16], mesActual[16], anoActual[16], Buffer[64]; int indice;  
  
    switch (message) {  
        case WM_INITDIALOG :  
            MessageBox(hDlg, TEXT("WM_INITDIALOG"), TEXT("DEBUG"), MB_OK);  
            InitDialog(hDlg, wParam, lParam); // iniciais dos Child Window Controls  
            return TRUE;  
  
        case WM_COMMAND :  
            switch ( LOWORD(wParam) ) {  
                case IDOK: EndDialog(hDlg, TRUE); return TRUE; // Close button message  
  
                case IDC_INSERTDATA: // Insert data button message  
                    SendMessage(GetDlgItem(hDlg, IDC_CBDIA), WM_GETTEXT, 12, (LPARAM)diaActual);  
                    SendMessage(GetDlgItem(hDlg, IDC_CBMES), WM_GETTEXT, 12, (LPARAM)mesActual);  
                    SendMessage(GetDlgItem(hDlg, IDC_CBANO), WM_GETTEXT, 12, (LPARAM)anoActual);  
                    _stprintf( Buffer, TEXT("%s-%s-%s"), diaActual, mesActual, anoActual);  
                    SendMessage(GetDlgItem(hDlg, IDC_EDIT1), WM_SETTEXT, 0, (LPARAM)Buffer);  
                    return TRUE;  
  
                . . .  
            }  
        }  
    }
```



Dialog Procedure (cont.)

```
. . .  
  
case IDC_LBDIASEM: // list box message  
    if ( HIWORD(wParam) == LBN_DBLCLK) {  
        indice = SendMessage(GetDlgItem(hDlg, IDC_LBDIASEM), LB_GETCURSEL, 0, 0);  
        SendMessage(GetDlgItem(hDlg, IDC_LBDIASEM), LB_GETTEXT, indice,  
                    (LPARAM)Buffer);  
        SetDlgItemText(hDlg, IDC_EDIT1, (LPCTSTR)Buffer);  
    };  
    return TRUE;  
case IDC_RADIO1: // radio button message  
    if ( HIWORD(wParam) == BN_CLICKED) {  
        bRadio = bRadio == BST_CHECKED ? BST_UNCHECKED : BST_CHECKED;  
        SendMessage(GetDlgItem(hDlg, IDC_RADIO1), BM_SETCHECK, (LPARAM) bRadio, 0);  
    };  
    return TRUE;  
case IDC_CHECK1: // Check button message  
    if ( HIWORD(wParam) == BN_CLICKED) {  
        bCheck = bCheck == BST_CHECKED ? BST_UNCHECKED : BST_CHECKED;  
        SendMessage(GetDlgItem(hDlg, IDC_CHECK1), BM_SETCHECK, (LPARAM) bCheck, 0);  
    };  
    return TRUE;  
} // switch ( LOWORD(wParam) )  
break;  
}; // switch (message)  
return FALSE; }
```



Init Dialog

```
static BOOL bCheck, bRadio;

void InitDialog(HWND hDlg, WPARAM wParam, LPARAM lParam) {

    TCHAR mesesAno[12][16]={ // Meses do ano
        TEXT("Janeiro"), TEXT("Fevereiro"),TEXT("Março"),
        TEXT("Abril"),   TEXT("Maio"),      TEXT("Junho"),
        TEXT("Julho"),   TEXT("Agosto"),   TEXT("Setembro"),
        TEXT("Outubro"), TEXT("Novembro"), TEXT("Dezembro")  };

    TCHAR diasSemana[7][16] = { // Dias da semana
        TEXT("Segunda"), TEXT("Terça"),    TEXT("Quarta"), TEXT("Quinta"),
        TEXT("Sexta"),   TEXT("Sabado"),   TEXT("Domingo")  };

    TCHAR dia[8], ano[8];

    SendMessage(GetDlgItem(hDlg, IDC_EDIT1), WM_SETTEXT, 0 ,
                (LPARAM)TEXT("Edite Aqui Texto"));

    bCheck = BST_CHECKED;
    SendMessage(GetDlgItem(hDlg, IDC_RADIO1), BM_SETCHECK, (LPARAM)bRadio, 0);

    bRadio = BST_CHECKED;
    SendMessage(GetDlgItem(hDlg, IDC_CHECK1), BM_SETCHECK, (LPARAM)bCheck, 0);
    . . .
```



Init Dialog (cont.)

```
. . .
for (int indiceDia=1; indiceDia<= 31; indiceDia++) {
    _stprintf( dia, TEXT("%2d"), indiceDia);
    SendMessage(GetDlgItem(hDlg, IDC_CBDIA), CB_ADDSTRING, 0, (LONG) (LPSTR)dia);
}
SendMessage(GetDlgItem(hDlg, IDC_CBDIA), CB_SETCURSEL, 0, 0);

for (int indiceMes=0; indiceMes<12; indiceMes++)
    SendMessage( GetDlgItem(hDlg, IDC_CBMES), CB_ADDSTRING, 0,
        (LONG)(LPSTR)mesesAno[indiceMes]);
SendMessage(GetDlgItem(hDlg, IDC_CBMES), CB_SETCURSEL, 0, 0);

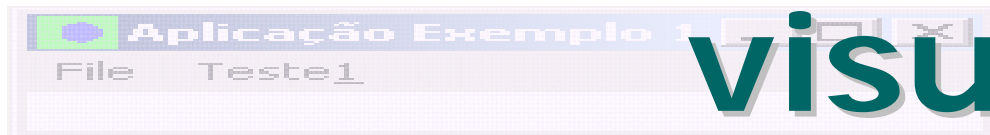
for (int indiceAno=0; indiceAno<6; indiceAno++) {
    _stprintf( ano, TEXT("%4d"), 2000+indiceAno);
    SendMessage(GetDlgItem(hDlg, IDC_CBANO), CB_ADDSTRING, 0, (LONG) (LPSTR)ano);
}
SendMessage(GetDlgItem(hDlg, IDC_CBANO), CB_SETCURSEL, 0, 0);

for (int indiceSem=0; indiceSem<7; indiceSem++)
    SendMessage(GetDlgItem(hDlg, IDC_LBDIASSEM), LB_ADDSTRING, 0,
        (LONG)(LPSTR)diasSemana[indiceSem]);
SendMessage(GetDlgItem(hDlg, IDC_LBDIASSEM), LB_SETCURSEL, 0, 0);
}
```



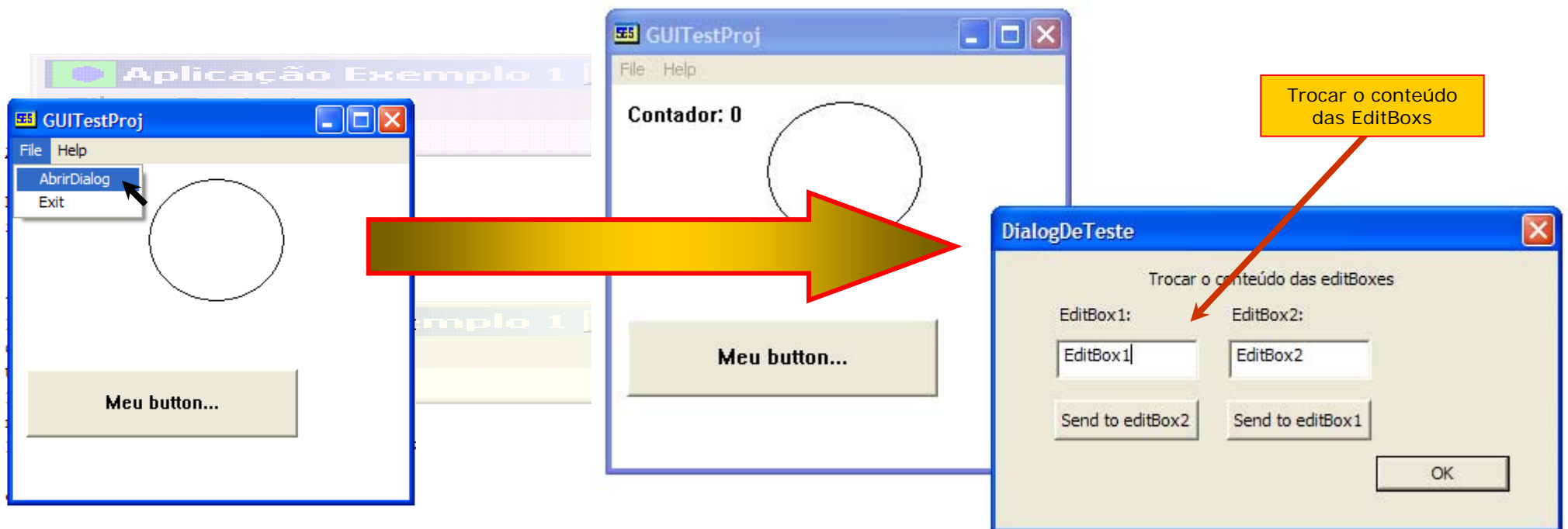


Definição de um Janela de diálogo utilizando o visual studio



Janelas de diálogo (Dialog Boxes - DB)

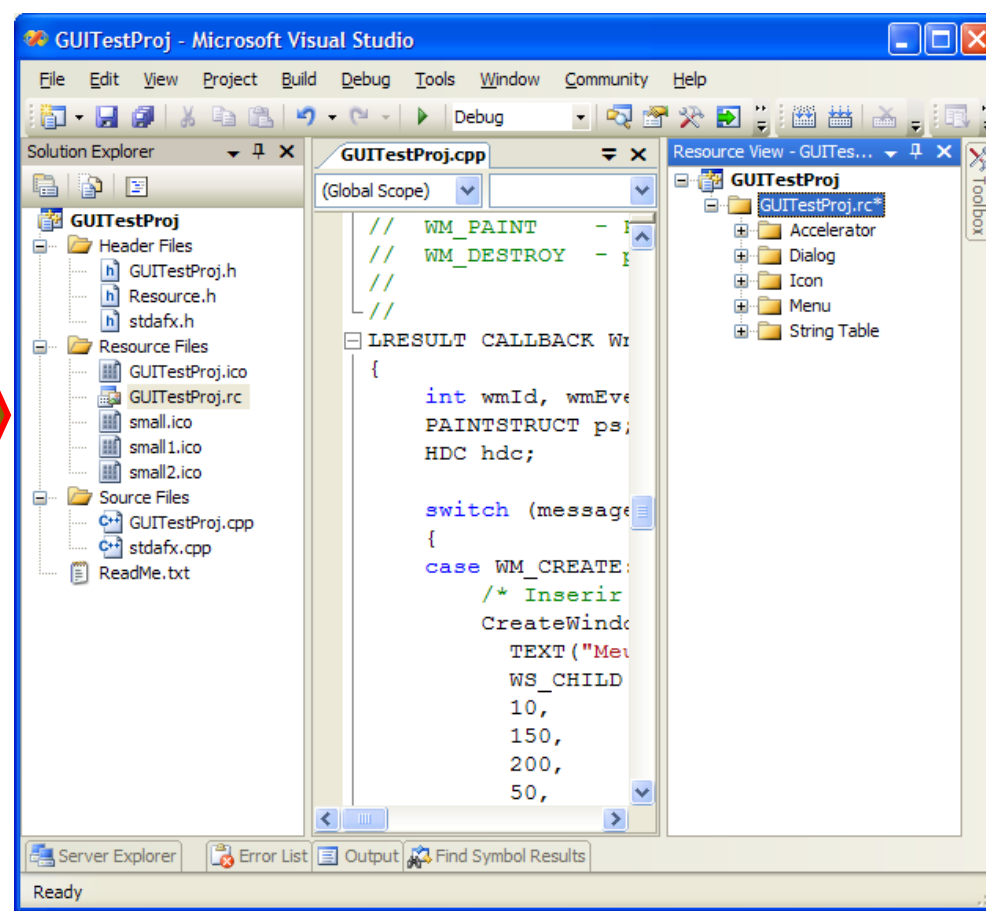
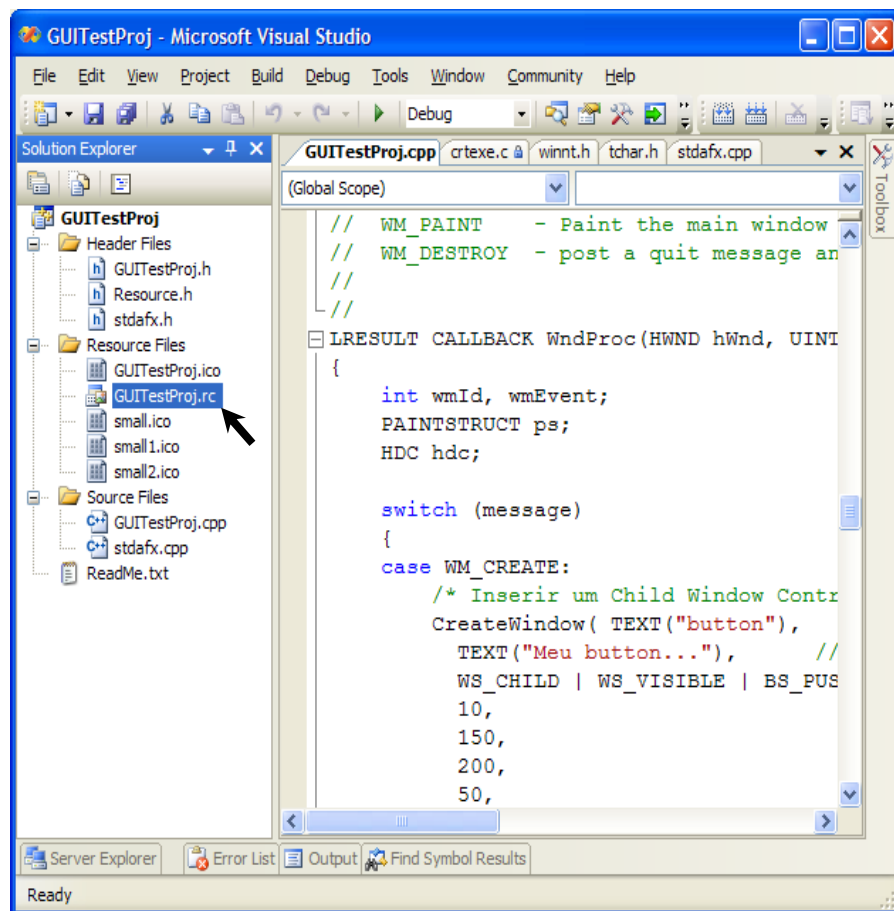
- Exemplo de criar uma DialogBox com o *resource editor*



Criar uma nova DialogBox



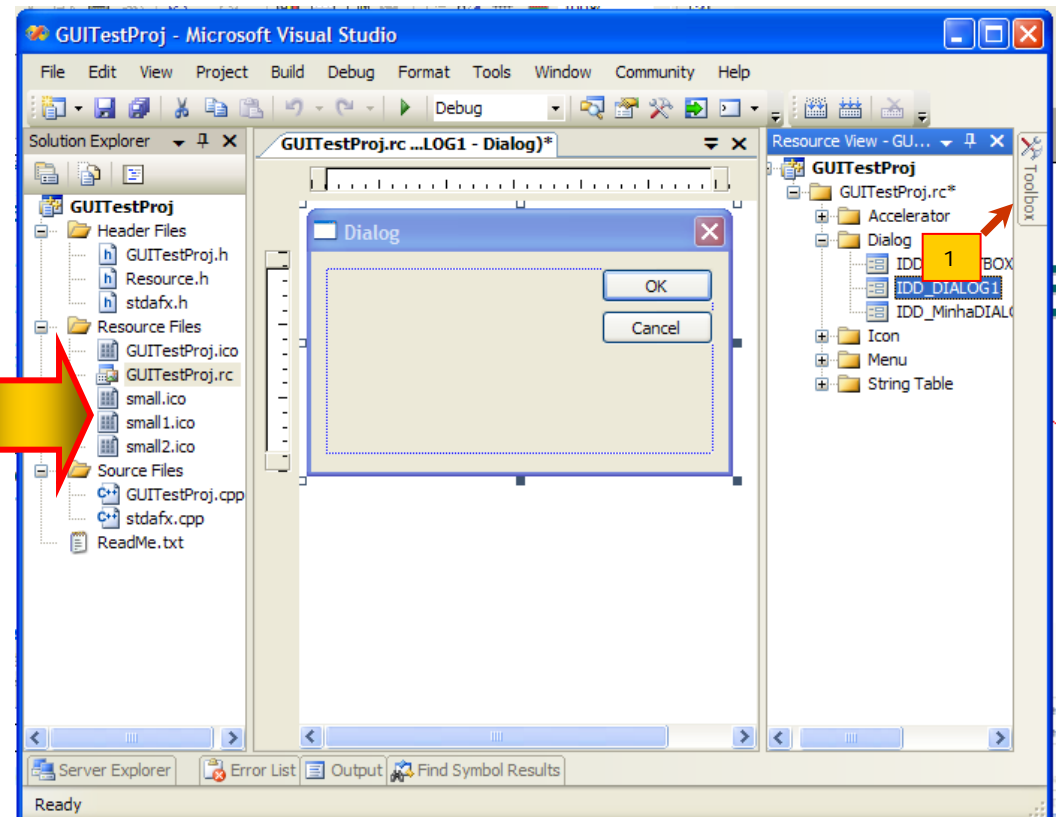
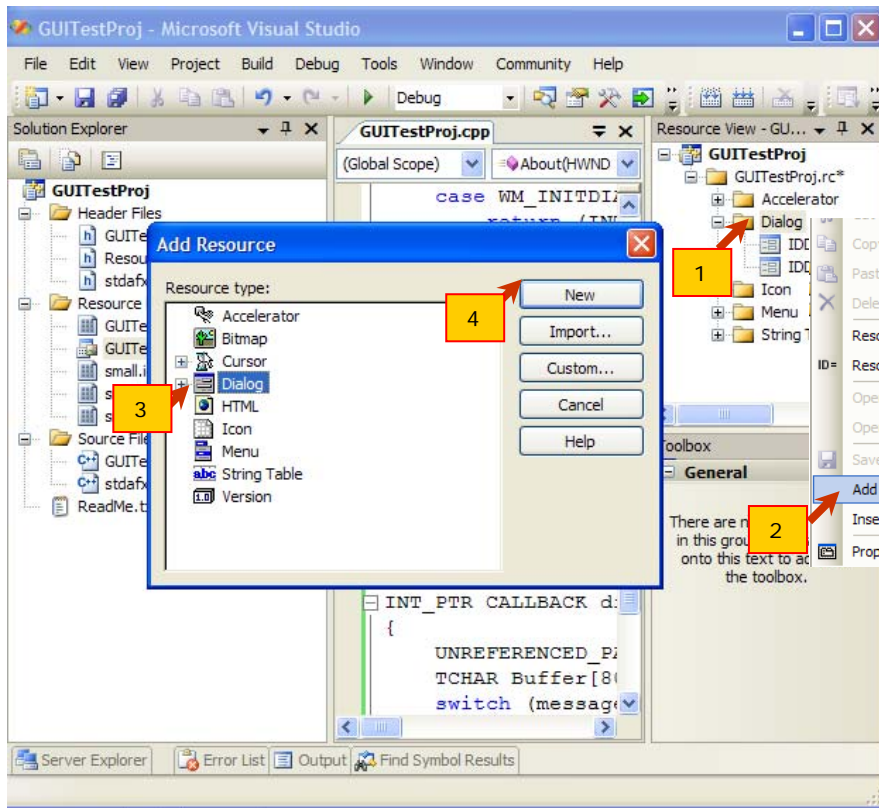
1º - Abrir o *Resource View*



Criar uma nova DialogBox

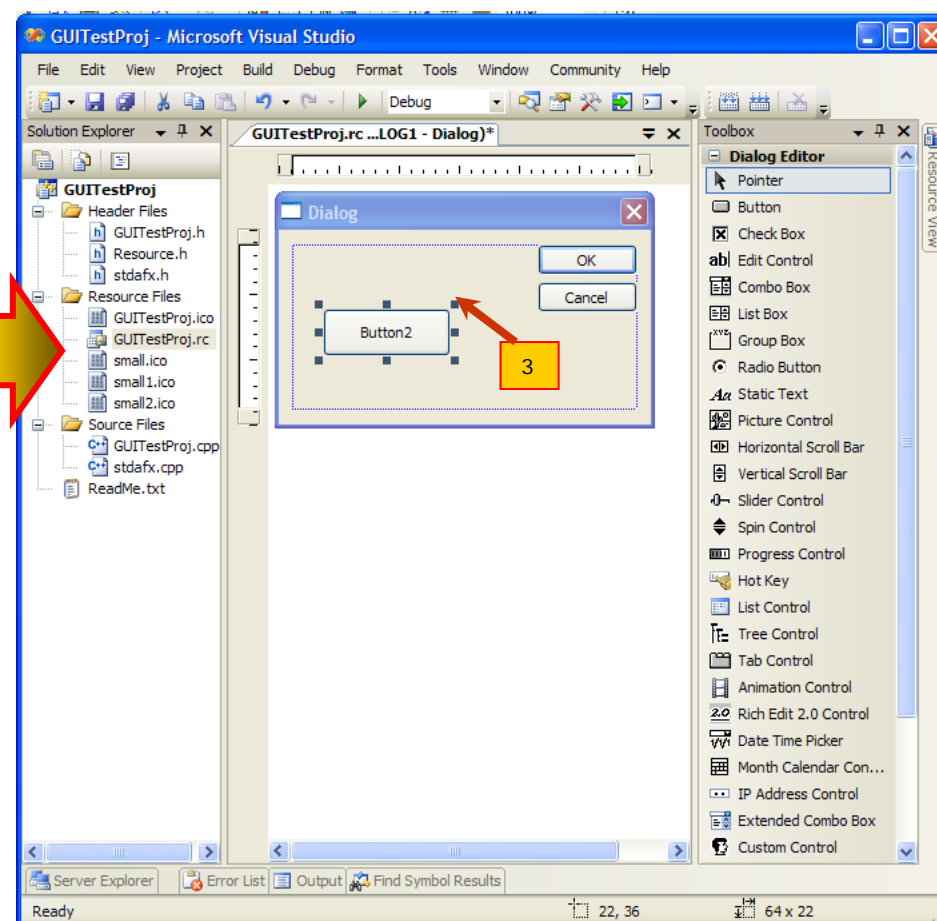
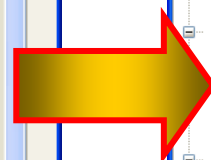
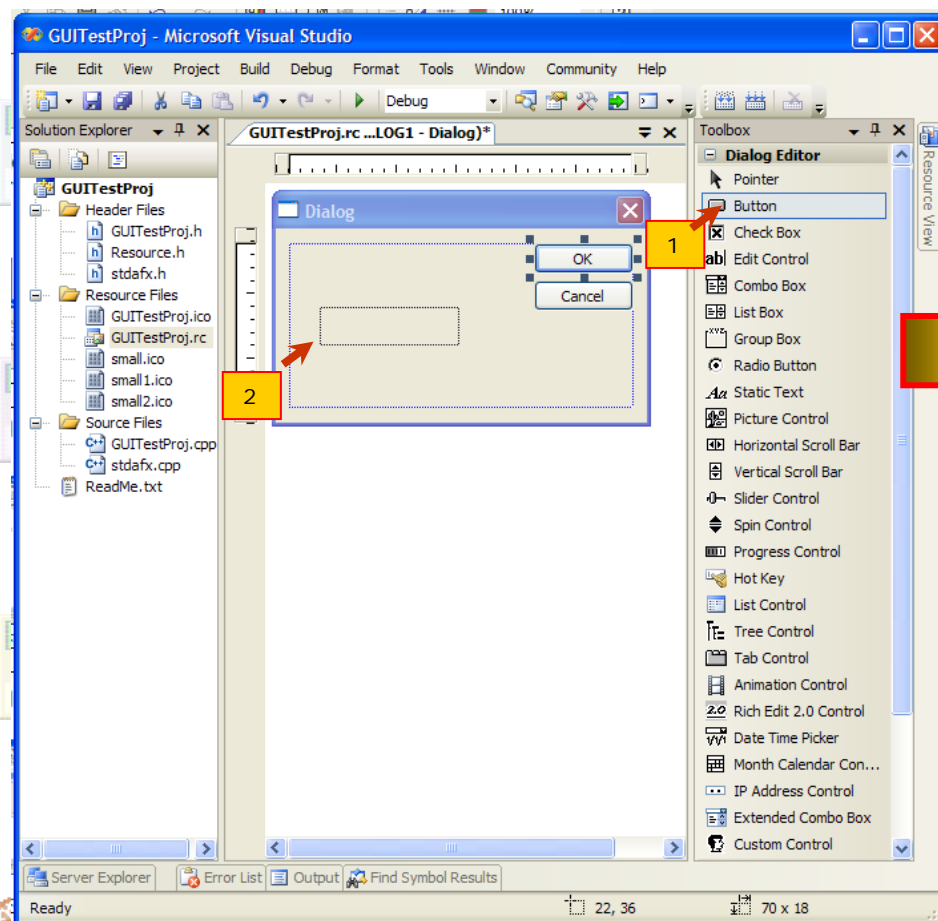


- 2º - Criar uma nova Dialog



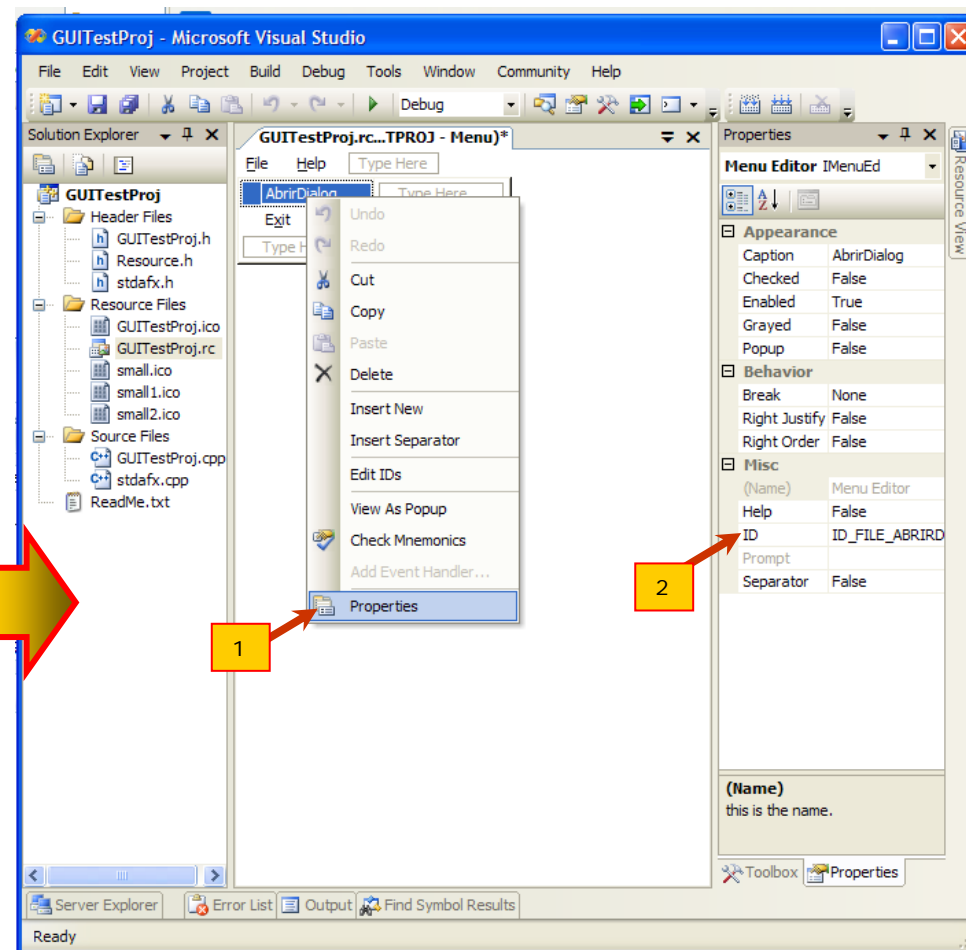
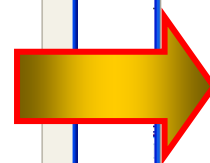
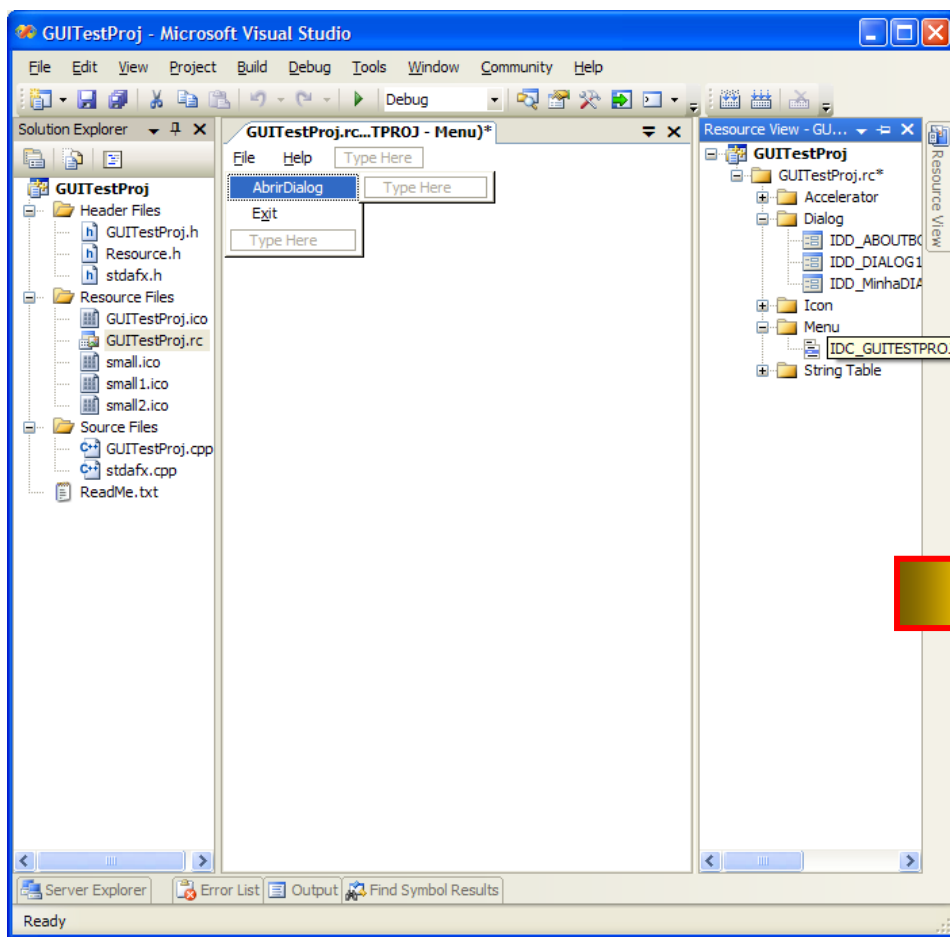
Criar uma nova DialogBox

- 3º - Utilizar a *ToolBox* para fazer *Drag&Drop* dos controlos que se pretendem.



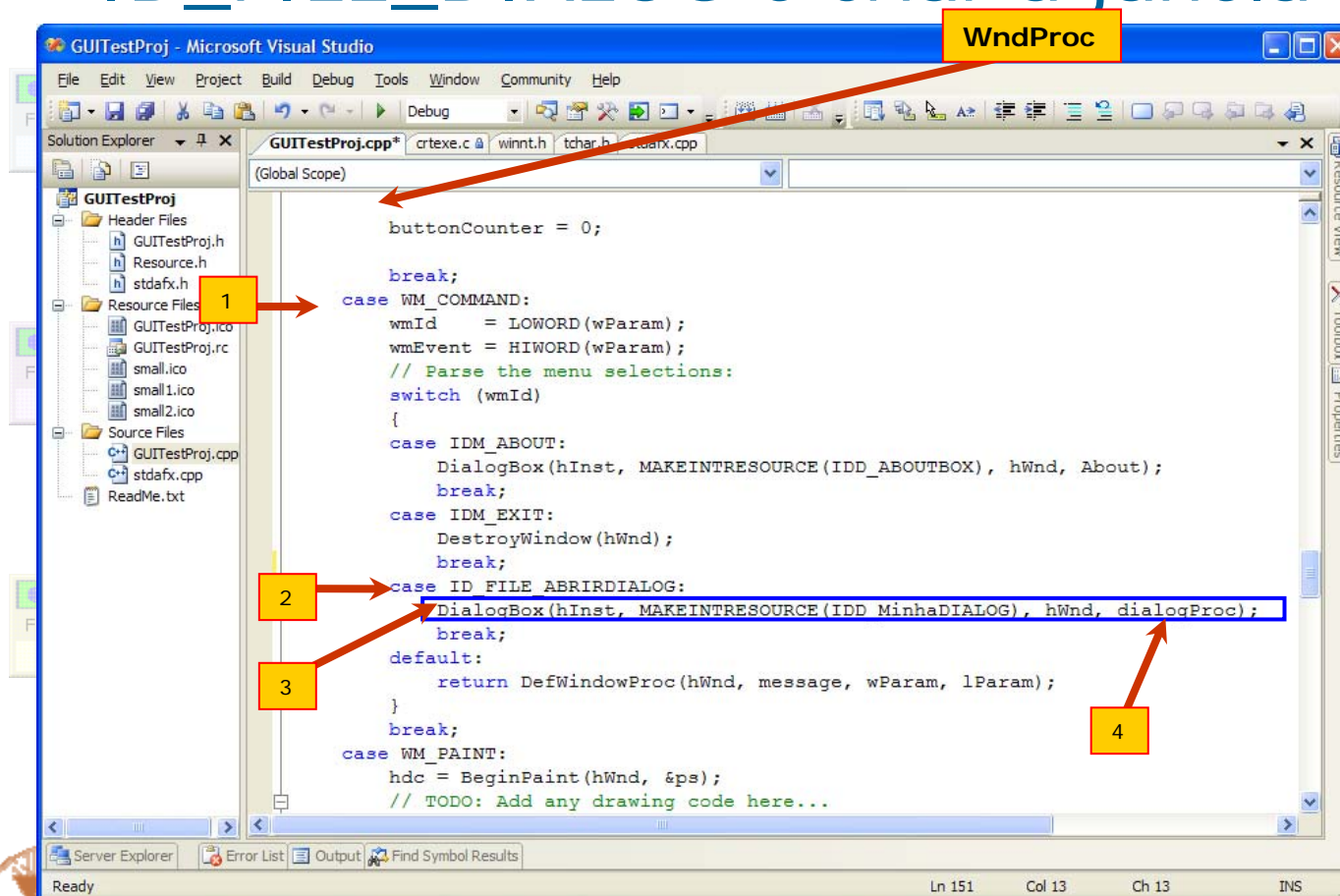
Criar uma nova *DialogBox*

- 4º - Criar no Menu da aplicação a opção para abrir a *dialog*.



Criar uma nova *DialogBox*

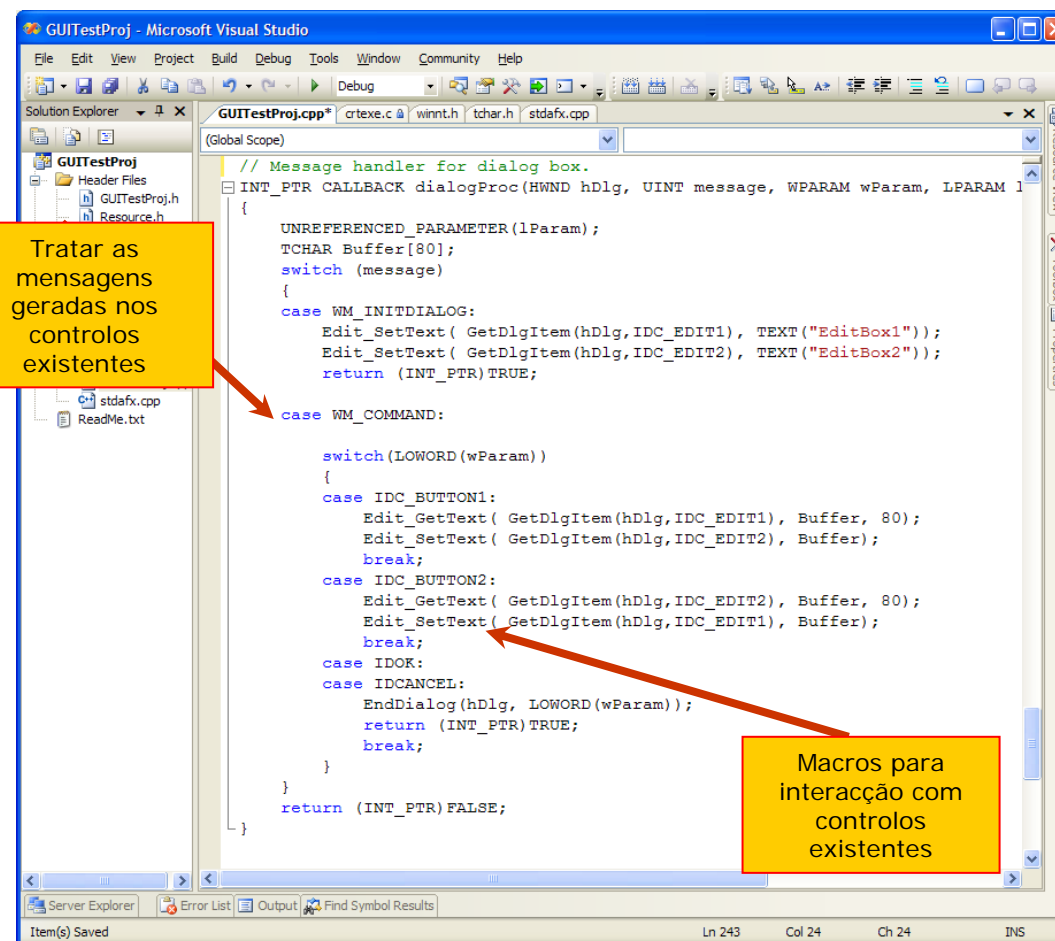
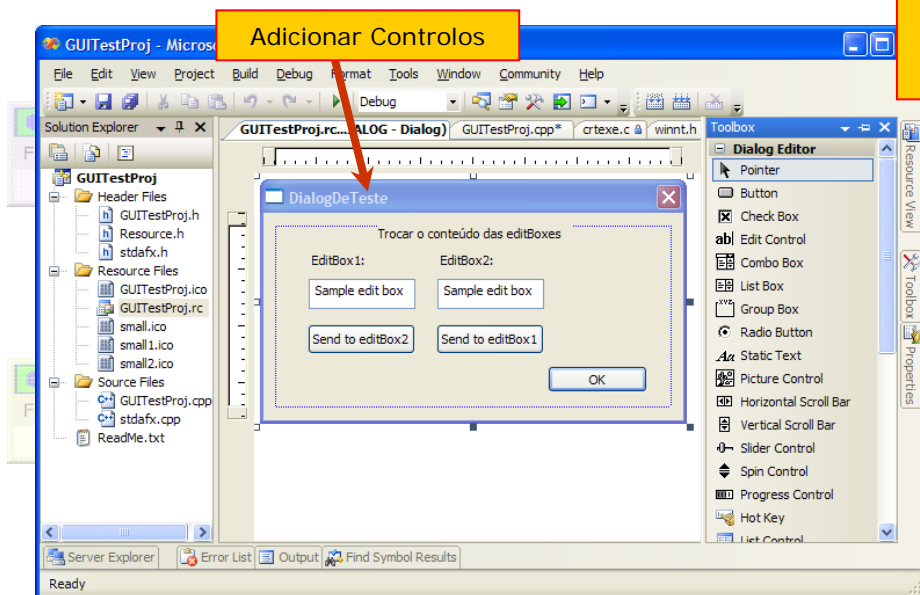
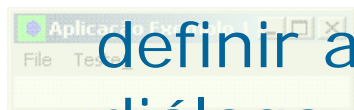
- 5º - Apanhar a mensagem que trata o evento de escolher a opção do menu com o id= ID_FILE_DIALOG e criar a janela de diálogo.



- 1 e 2: Acrescentar na mensagem WM_COMMAND o tratamento de pressionar o controlo ID_FILE_ABRIRDIALOG
- 3: Criar a *DialogBox*
 - Bloqueante enquanto a janela de diálogo não terminar
- 4: Registrar qual a função que trata as mensagens geradas para a janela de diálogo (*dialogProc*)

Criar uma nova *DialogBox*

- 6º - Adicionar os controlos à janela de diálogo e definir a função que trata os eventos da janela de diálogo (*dialogProc*).

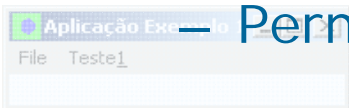


Mais informação em anexo



- Anexo 1:

- Message Crackers (Macros para tratamento de mensagens)



- Permite simplificar a escrita e a legibilidade do código

- Anexo 2:

- Janelas de Diálogo



- *Child Window Controls*: Controlos já existentes (botões, etc...)
- Macros para facilitar a interacção com os *CWC*.



- Anexo 3:

- Desenho em janelas de diálogo

