Урок №4

Содержание

1. Создание кнопок с помощью функции CreateWindowEx.
2. Элемент управления «текстовое поле ввода» (Edit Control).
3. Сообщения текстовых полей ввода.
4. Распаковщики сообщений.

1. Создание кнопок с помощью функции CreateWindowEx

В предыдущем уроке был рассмотрен способ создания кнопок с использованием средств интегрированной среды разработки приложений **Microsoft Visual Studio**. Существует альтернативный способ создания кнопки - использование функции **CreateWindowEx**. В этом случае во втором параметре функции передается имя предопределенного оконного класса – **BUTTON**.

В качестве примера, демонстрирующего программное создание кнопок, рассмотрим следующее приложение, в котором при нажатии на кнопку «Старт» начинается «Слайд-шоу» (периодическая смена изображений), а при нажатии на кнопку «Стоп» «Слайд-шоу» останавливается (исходный код приложения находится в папке **SOURCE/Button_CreateWindowEx**).

```
#include <windows.h>
#include "resource.h"

#define LEFT_START 52
#define TOP_START 100
```

```
#define WIDTH_START 76
#define HEIGHT_START 30
#define LEFT_STOP 168
#define TOP_STOP 100
#define WIDTH_STOP 76
#define HEIGHT_STOP 30
#define LEFT_PICTURE 100
#define TOP_PICTURE 5
#define WIDTH_PICTURE 86
#define HEIGHT_PICTURE 86

HWND hStart, hStop, hPicture;
HBITMAP hBmp[5];
HINSTANCE hInst;

BOOL CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                        LPSTR lpszCmdLine, int nCmdShow)
{
      hInst = hInstance;
      return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                        DlgProc);
}

BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
      switch(message)
      {
            case WM_CLOSE:
                  EndDialog(hWnd, 0);
                  return TRUE;

            case WM_INITDIALOG:
                  hStart = CreateWindowEx(WS_EX_DLGMODALFRAME, TEXT("BUTTON"),
                              TEXT("Start"), WS_CHILD | WS_VISIBLE,
                              LEFT_START, TOP_START, WIDTH_START,
                              HEIGHT_START, hWnd, 0, hInst, 0);

                  hStop = CreateWindowEx(WS_EX_DLGMODALFRAME, TEXT("BUTTON"),
                              TEXT("Stop"), WS_CHILD | WS_VISIBLE |
                              WS_DISABLED, LEFT_STOP, TOP_STOP, WIDTH_STOP,
                              HEIGHT_STOP, hWnd, 0, hInst, 0);

                  hPicture = CreateWindowEx(0, TEXT("BUTTON"), 0,
                              WS_CHILD | WS_VISIBLE | BS_BITMAP,
                              LEFT_PICTURE, TOP_PICTURE, WIDTH_PICTURE,
                              HEIGHT_PICTURE, hWnd, 0, hInst, 0);
                  for(int i = 0; i < 5; i++)
                        hBmp[i] = LoadBitmap(hInst,
                              MAKEINTRESOURCE(IDB_BITMAP1 + i));
                  SendMessage(hPicture, BM_SETIMAGE, WPARAM(IMAGE_BITMAP),
                              LPARAM(hBmp[0]));
                  return TRUE;

            case WM_COMMAND:
                  {
                        HWND h = GetFocus();
```

```
                        TCHAR text[10];
                        GetWindowText(h, text, 10);
                        if(!lstrcmp(text, TEXT("Start")))
                        {
                              SetTimer(hWnd, 1, 1000, 0);
                              EnableWindow(hStart, 0);
                              EnableWindow(hStop, 1);
                              SetFocus(hStop);
                        }
                        else if(!lstrcmp(text, TEXT("Stop")))
                        {
                              KillTimer(hWnd, 1);
                              EnableWindow(hStart, 1);
                              EnableWindow(hStop, 0);
                              SetFocus(hStart);
                        }
                  }
                  return TRUE;

            case WM_TIMER:
                  static int index = 0;
                  index++;
                  if(index > 4)
                        index = 0;
                  SendMessage(hPicture, BM_SETIMAGE, WPARAM(IMAGE_BITMAP),
                              LPARAM(hBmp[index]));
                  return TRUE;
      }
      return FALSE;
}
```

Анализируя вышеприведенный код, отметим некоторые стили, которые использовались при создании кнопок.

- Стиль **WS_CHILD** позволяет создать кнопку как дочернее окно диалога.
- Стиль **WS_VISIBLE** управляет видимостью кнопки.
- Стиль **BS_BITMAP** указывает на то, что на кнопке должен быть рисунок (растровый битовый образ) вместо текста.
- Стиль **WS_DISABLED** указывает на то, что кнопка будет запрещённой. Как известно, запрещённые элементы выводятся на экран серым цветом и не воспринимают пользовательский ввод с клавиатуры или от мыши.

Следующий фрагмент кода демонстрирует программный способ создания флажка и группы переключателей:

```
HWND hCheck = CreateWindowEx(0, TEXT("BUTTON"),TEXT("CheckBox"),
            WS_CHILD | WS_VISIBLE | BS_AUTOCHECKBOX,
            LEFT, TOP, WIDTH, HEIGHT, hWnd, 0, GetModuleHandle(0), 0);
HWND hRadio[5];
for(int i = 0; i < 5; i++)
{
      hRadio[i] = CreateWindowEx(0, TEXT("BUTTON"), TEXT("RadioButton"),
                        WS_CHILD | WS_VISIBLE |  BS_AUTORADIOBUTTON,
                        LEFT, TOP + i*20, WIDTH, HEIGHT, hWnd, 0,
                        GetModuleHandle(0), 0);
}
```

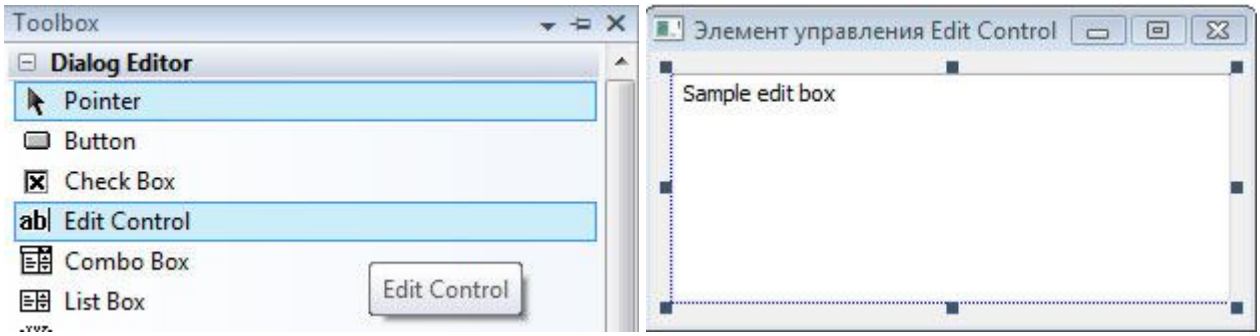
2. Элемент управления «текстовое поле ввода» (Edit Control)

Текстовое поле ввода (или окно редактирования) представляет собой прямоугольное окно, в которое можно вводить текст с клавиатуры. На практике применяются различные текстовые поля ввода в самом широком спектре: от небольшого однострочного поля ввода до многострочного элемента управления с автоматическим переносом строк, как в программе **Microsoft Notepad**.

Создать текстовое поле ввода на форме диалога можно двумя способами:

- с помощью средств интегрированной среды разработки приложений **Microsoft Visual Studio**;
- посредством вызова функции **CreateWindowEx**.

При первом способе необходимо определить **Edit Control** в шаблоне диалогового окна на языке описания шаблона диалога. Это произойдёт автоматически, если активизировать окно **Toolbox** (<**Ctrl**><**Alt**><**X**>) и «перетащить» текстовое поле ввода на форму диалога.



После размещения текстового поля ввода на форме диалога ему назначается идентификатор (например, **IDC_EDIT1**), который впоследствии можно изменить на другой идентификатор, отражающий семантику ресурса.

Рассмотрим некоторые свойства текстового поля ввода.

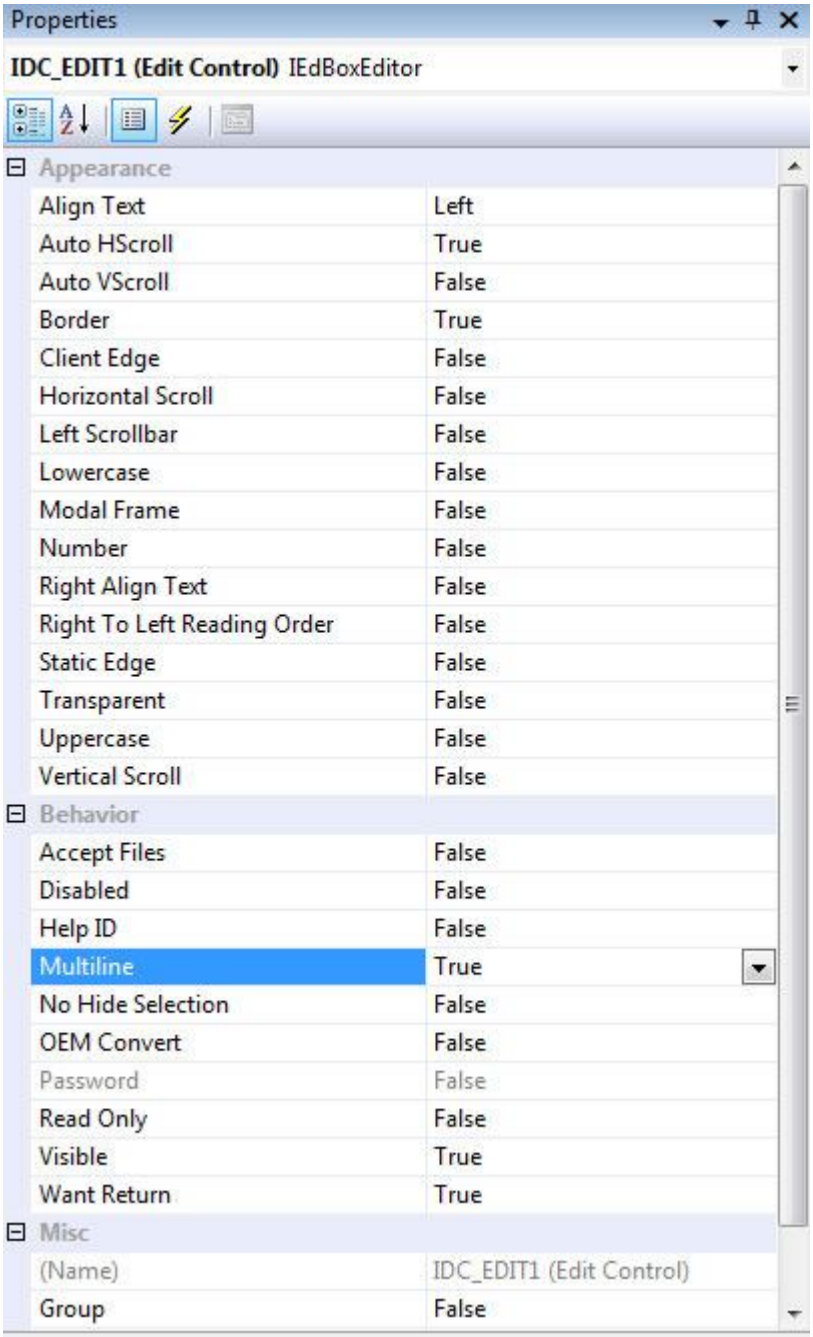
По умолчанию **Edit Control** является однострочным, с автоматической горизонтальной прокруткой (свойство **Auto HScroll**), с объемной рамкой (свойство **Border**) и выравниванием текста по левой границе окна (свойство **Align Text** со значением **Left**).

Если установить истинным значение свойства **Multiline**, то текстовое поле ввода будет работать в многострочном режиме. При этом станут доступными для использования свойства **Horizontal Scroll**, **Vertical Scroll**, **Auto VScroll**.

Значение **True** свойства **Number** разрешает вводить в **Edit Control** только цифры. Если же окно редактирования предназначено для ввода пароля, то следует установить истинным значение свойства **Password**, и тогда вместо вводимых символов в поле ввода будут отображаться символы звездочки.

Свойство **No Hide Selection** позволяет элементу управления всегда показывать выделенную подстроку, даже если **Edit Control** теряет фокус.

Свойство **Want Return** используется для многострочного окна редактирования. Если для этого свойства установлено значение **True**, то нажатие клавиши **<Enter>** приводит к вводу символа возврата каретки.



Взаимоисключающие свойства **Uppercase** и **Lowercase** преобразуют вводимые символы. В первом случае происходит конвертирование в символы верхнего регистра, а во втором — в символы нижнего регистра.

Истинное значение свойства **Read Only** устанавливает для **Edit Control** режим «только для чтения», не позволяющий пользователю вводить или редактировать текст.

Следующий пример демонстрирует использование текстовых полей ввода (исходный код приложения находится в папке **SOURCE/ Using Edit Control (IDE)**).

```
#include <windows.h>
#include "resource.h"

HWND hLogin, hPassword;

BOOL CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                   LPSTR lpszCmdLine, int nCmdShow)
{
      return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                        DlgProc);
}

BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
      switch(message)
      {
            case WM_CLOSE:
                  EndDialog(hWnd, 0);
                  return TRUE;

            case WM_INITDIALOG:
                  hLogin = GetDlgItem(hWnd, IDC_LOGIN);
                  hPassword = GetDlgItem(hWnd, IDC_PASSWORD);
                  return TRUE;

            case WM_COMMAND:
                  if(LOWORD(wParam) == IDC_ENTRY)
                  {
                        TCHAR text[100], login[20], password[20];
                        GetWindowText(hLogin, login, 20);
                        GetWindowText(hPassword, password, 20);
                        if(lstrlen(login) == 0 || lstrlen(password) == 0)
                              MessageBox(hWnd,
                              TEXT("Не введён логин или пароль!"),
                              TEXT("Авторизация"), MB_OK | MB_ICONSTOP);
                        else
                        {
                              wsprintf(text,
                              TEXT("Логин: %s\nПароль: %s"), login, password);
                              MessageBox(hWnd, text, TEXT("Авторизация"),
                                          MB_OK | MB_ICONINFORMATION);
                        }
                        SetWindowText(hLogin, 0);
                        SetWindowText(hPassword, 0);
                        SetFocus(hLogin);
                  }
                  return TRUE;
      }
      return FALSE;
}
```

Альтернативный способ создания текстового поля ввода - использование функции **CreateWindowEx**. В этом случае во втором параметре функции передается имя предопределенного оконного класса – **EDIT**.

Следующий пример демонстрирует программный способ создания текстовых полей ввода (исходный код приложения находится в папке **SOURCE/ Using Edit Control CreateWindowEx**).

```
#include <windows.h>
#include "resource.h"

HWND hEdit1, hEdit2, hButton;
HINSTANCE hInst;

BOOL CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                        LPSTR lpszCmdLine, int nCmdShow)
{
      hInst = hInstance;
      return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                        DlgProc);
}

BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{

      switch(message)
      {
            case WM_CLOSE:
                  EndDialog(hWnd, 0);
                  return TRUE;

            case WM_INITDIALOG:
                  hEdit1 = CreateWindowEx(WS_EX_CLIENTEDGE, TEXT("EDIT"), 0,
                        WS_CHILD | WS_VISIBLE | WS_VSCROLL | WS_TABSTOP |
                        ES_WANTRETURN | ES_MULTILINE | ES_AUTOVSCROLL,
                        10, 7, 150, 100, hWnd, 0, hInst, 0);
                  hEdit2 = CreateWindowEx(WS_EX_CLIENTEDGE, TEXT("EDIT"), 0,
                        WS_CHILD | WS_VISIBLE | WS_VSCROLL | ES_WANTRETURN |
                        ES_MULTILINE | ES_AUTOVSCROLL | ES_READONLY,
                        170, 7, 150, 100, hWnd, 0, hInst, 0);
                  hButton = CreateWindowEx(WS_EX_CLIENTEDGE |
                        WS_EX_DLGMODALFRAME, TEXT("BUTTON"),
                        TEXT("Click Me!"), WS_CHILD | WS_VISIBLE | WS_TABSTOP,
                        10, 110, 310, 40, hWnd, 0, hInst, 0);
                  return TRUE;

            case WM_COMMAND:
                  {
                        HWND h = (HWND) lParam;
                        if(h == hButton)
                        {
```

```
                                int length = SendMessage(hEdit1,
                                                WM_GETTEXTLENGTH, 0, 0);
                                TCHAR *buffer = new TCHAR[length + 1];
                                memset(buffer, 0, length + 1);
                                GetWindowText(hEdit1, buffer, length + 1);
                                SetWindowText(hEdit2, buffer);
                                delete [] buffer;
                            }
                        }
                        return TRUE;
                }
                return FALSE;
}
```

Анализируя вышеприведенный код, следует отметить сообщение **WM_GETTEXTLENGTH**, которое отправляется текстовому полю ввода с целью определения длины введённого текста.

Существует возможность изменения стилей элементов управления. Для этой цели используется функция API **SetWindowLong**:

```
LONG SetWindowLong(
    HWND hWnd, // дескриптор окна
    int nIndex, // индекс значения, которое нужно изменить
    LONG dwNewLong // новое значение
);
```

Функция API **GetWindowLong** позволяет получить текущие стили окна (элемента управления):

```
LONG GetWindowLong(
    HWND hWnd, // дескриптор окна
    int nIndex // индекс значения, которое нужно выбрать
);
```

Следующий пример демонстрирует программное изменение стилей кнопки, статика и текстового поля (исходный код приложения находится в папке **SOURCE/ ChangeStyles**).

```
#include <windows.h>
#include "resource.h"

HWND hButton, hEdit, hStatic;

BOOL CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                   LPSTR lpszCmdLine, int nCmdShow)
{
      return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                         DlgProc);
}

BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
      switch(message)
      {
            case WM_CLOSE:
                  EndDialog(hWnd, 0);
                  return TRUE;

            case WM_INITDIALOG:
                  hButton = GetDlgItem(hWnd, IDC_BUTTON1);
                  hEdit = GetDlgItem(hWnd, IDC_EDIT1);
                  hStatic = GetDlgItem(hWnd, IDC_STATIC1);
                  return TRUE;

            case WM_COMMAND:
                  if(LOWORD(wParam) == IDC_BUTTON2)
                  {
                        LONG styles = GetWindowLong(hButton, GWL_STYLE);
                        SetWindowLong(hButton, GWL_EXSTYLE, styles |
                              WS_EX_CLIENTEDGE | WS_EX_DLGMODALFRAME);
                        InvalidateRect(hButton, 0, 1);
                  }
                  else if(LOWORD(wParam) == IDC_BUTTON3)
                  {
                        LONG styles = GetWindowLong(hEdit, GWL_STYLE);
                        SetWindowLong(hEdit, GWL_STYLE, styles | ES_RIGHT);
                        InvalidateRect(hEdit, 0, 1);
                  }
                  else if(LOWORD(wParam) == IDC_BUTTON4)
                  {
                        LONG styles = GetWindowLong(hEdit, GWL_STYLE);
                        SetWindowLong(hStatic, GWL_EXSTYLE, styles |
                              WS_EX_CLIENTEDGE | WS_EX_DLGMODALFRAME);
                        InvalidateRect(hStatic, 0, 1);
                  }
                  return TRUE;
      }
      return FALSE;
}
```

3. Сообщения текстовых полей ввода

Для выполнения различных операций по редактированию текста приложение может отправлять следующие сообщения элементу управления **Edit Control**:

Код сообщения	wParam	lParam	Описание
EM_SETSEL	iStart	iEnd	Выделить текст, начиная с позиции **iStart** и заканчивая позицией **iEnd**
EM_GETSEL	&iStart	&iEnd	Получить начальное и конечное положения текущего выделения
WM_CLEAR	0	0	Удалить выделенный текст
WM_CUT	0	0	Удалить выделенный текст и поместить его в буфер обмена
WM_COPY	0	0	Скопировать выделенный текст в буфер обмена **Windows**
WM_PASTE	0	0	Вставить текст из буфера обмена в месте, соответствующем позиции курсора
EM_CANUNDO	0	0	Определить возможность отмены последнего действия
WM_UNDO	0	0	Отменить последнее действие
WM_GETTEXT	nMax	szBuffer	Скопировать текст (не более **nMax** символов) из элемента управления в буфер **szBuffer**
EM_GETLINECOUNT	0	0	Получить число строк для многострочного окна редактирования
EM_LINELENGTH	iLine	0	Получить длину строки **iLine**
EM_GETLINE	iLine	szBuffer	Скопировать строку **iLine** в буфер **szBuffer**
EM_LINEFROMCHAR	-1	0	Получить номер строки, в которой расположен курсор
EM_LINEINDEX	iLine	0	Получить номер первого символа строки **iLine**

Как известно, при воздействии на элемент управления диалога (например, при вводе текста в **Edit Control**) в диалоговую процедуру

DlgProc поступает сообщение **WM_COMMAND,** в котором **LOWORD(wParam)** содержит идентификатор элемента управления, **HIWORD(wParam)** содержит код уведомления (например, **EN_CHANGE)**, а **lParam** – дескриптор элемента управления.

Некоторые из кодов уведомления от текстового поля ввода приведены в таблице.

Код уведомления	Интерпретация
EN_SETFOCUS	Окно получило фокус ввода
EN_KILLFOCUS	Окно потеряло фокус ввода
EN_UPDATE	Содержимое окна будет меняться
EN_CHANGE	Содержимое окна изменилось
EN ERRSPACE	Произошло переполнение буфера редактирования

Для демонстрации обработки сообщения **WM_COMMAND** с кодом уведомления **EN_CHANGE** модифицируем пример, рассмотренный ранее (исходный код приложения находится в папке **SOURCE/ Autorization (EN_CHANGE)**).

```
#include <windows.h>
#include "resource.h"

HWND hLogin, hPassword, hEntry;
TCHAR text[100], login[20], password[20];

BOOL CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                   LPSTR lpszCmdLine, int nCmdShow)
{
      return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                       DlgProc);
}


BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{

      switch(message)
      {
            case WM_CLOSE:
                  EndDialog(hWnd, 0);
                  return TRUE;
```

```
            case WM_INITDIALOG:
                  hLogin = GetDlgItem(hWnd, IDC_LOGIN);
                  hPassword = GetDlgItem(hWnd, IDC_PASSWORD);
                  hEntry = GetDlgItem(hWnd, IDC_ENTRY);
                  return TRUE;

            case WM_COMMAND:
                  if((LOWORD(wParam) == IDC_LOGIN ||
                  LOWORD(wParam) == IDC_PASSWORD) &&
                  HIWORD(wParam) == EN_CHANGE)
                  {
                        GetWindowText(hLogin, login, 20);
                        GetWindowText(hPassword, password, 20);
                        if(lstrlen(login) == 0 || lstrlen(password) < 6)
                              EnableWindow(hEntry, 0);
                        else
                              EnableWindow(hEntry, 1);
                  }
                  if(LOWORD(wParam) == IDC_ENTRY)
                  {
                        wsprintf(text, TEXT("Логин: %s\nПароль: %s"),
                                    login, password);
                        MessageBox(hWnd, text, TEXT("Авторизация"),
                                    MB_OK | MB_ICONINFORMATION);
                        SetWindowText(hLogin, 0);
                        SetWindowText(hPassword, 0);
                        SetFocus(hLogin);
                        EnableWindow(hEntry, 0);
                  }
                  return TRUE;
      }
      return FALSE;
}
```

В следующем примере рассматриваются сообщения, отправляемые текстовому полю ввода, при выполнении различных операций по редактированию текста (исходный код приложения находится в папке **SOURCE / EditControlMessages**).

```
#include <windows.h>
#include "resource.h"

#define WM_POSITION WM_APP // Пользовательское сообщение

HWND hEdit, hCopy, hCut, hPaste, hDelete, hUndo, hSelectAll;
WNDPROC OriginalProc = NULL;

BOOL CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                  LPSTR lpszCmdLine, int nCmdShow)
{
```

```
	return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
			DlgProc);
}

void GetPosition()
{
	// Получим номер строки, в которой расположен курсор
	int row = SendMessage(hEdit, EM_LINEFROMCHAR, -1, 0);
	DWORD Start, End;

	// Получим границы выделения текста
	SendMessage(hEdit, EM_GETSEL, (WPARAM)&Start, (LPARAM)&End);

	// Получим номер первого символа указанной строки
	int col = Start - SendMessage(hEdit, EM_LINEINDEX, row, 0);

	// Получим дескриптор родительского окна (диалога)
	HWND hParent = GetParent(hEdit);

	// Установим в заголовок главного окна текущие координаты курсора
	TCHAR buffer[50] = {0};
	wsprintf(buffer, TEXT("Строка %d, Столбец %d"), row+1, col+1);
	SetWindowText(hParent, buffer);
	SetFocus(hEdit); // Переведём фокус ввода на Edit Control
}


void EnableDisableButton()
{
	// Получим границы выделения текста
	DWORD dwPosition = SendMessage(hEdit, EM_GETSEL, 0, 0);
	WORD wBeginPosition = LOWORD(dwPosition);
	WORD wEndPosition = HIWORD(dwPosition);

	if(wEndPosition != wBeginPosition) // Выделен ли текст?
	{
		EnableWindow(hCopy, 1);
		EnableWindow(hCut, 1);
		EnableWindow(hDelete, 1);
	}
	else
	{
		EnableWindow(hCopy, 0);
		EnableWindow(hCut, 0);
		EnableWindow(hDelete, 0);
	}

	// Имеется ли текст в буфере обмена?
	if(IsClipboardFormatAvailable(CF_TEXT))
		EnableWindow(hPaste, 1);
	else
		EnableWindow(hPaste, 0);

	// Существует ли возможность отмены последнего действия?
	if(SendMessage(hEdit, EM_CANUNDO, 0, 0))
		EnableWindow(hUndo, 1);
	else
		EnableWindow(hUndo, 0);
```

```
	// Определим длину текста в Edit Control
	int length = SendMessage(hEdit, WM_GETTEXTLENGTH, 0, 0);

	// Выделен ли весь текст в Edit Control?
	if(length != wEndPosition - wBeginPosition)
		EnableWindow(hSelectAll, 1);
	else
		EnableWindow(hSelectAll, 0);
}


LRESULT CALLBACK EditProc(HWND hWnd, UINT message, WPARAM wParam,
				LPARAM lParam)
{
	if(	message == WM_POSITION || message == WM_LBUTTONDOWN ||
		message == WM_LBUTTONUP || message == WM_KEYDOWN ||
		message == WM_KEYUP || message == WM_MOUSEMOVE &&
		(wParam & MK_LBUTTON))
	{
		EnableDisableButton();
		GetPosition();
	}
	// Вызов стандартного обработчика сообщений
	return CallWindowProc(OriginalProc, hWnd, message, wParam, lParam);
}

BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	switch(message)
	{
		case WM_CLOSE:
			EndDialog(hWnd, 0);
			return TRUE;

		case WM_INITDIALOG:
			hEdit = GetDlgItem(hWnd, IDC_EDIT1);
			hCopy = GetDlgItem(hWnd, ID_COPY);
			hCut = GetDlgItem(hWnd, ID_CUT);
			hPaste = GetDlgItem(hWnd, ID_PASTE);
			hDelete = GetDlgItem(hWnd, ID_DELETE);
			hSelectAll = GetDlgItem(hWnd, ID_SELECTALL);
			hUndo = GetDlgItem(hWnd, ID_UNDO);

			// переопределим оконную процедуру текстового поля
			OriginalProc = (WNDPROC) SetWindowLong(hEdit, GWL_WNDPROC,
					LONG(EditProc));

			/* отправка пользовательского сообщения для определения
				текущих координат курсора */
			SendMessage(hEdit, WM_POSITION, 0, 0);
			return TRUE;

		case WM_COMMAND:
			switch(LOWORD(wParam))
			{
				case ID_UNDO:
					// отменить последнее действие
					SendMessage(hEdit, WM_UNDO, 0, 0);
```

```
                              // определить текущие координаты курсора
                              SendMessage(hEdit, WM_POSITION, 0, 0);
                              break;
                        case ID_COPY:
                              // скопировать выделенный фрагмент текста
                              SendMessage(hEdit, WM_COPY, 0, 0);
                              break;
                        case ID_PASTE:
                              // вставить текст в Edit из буфера обмена
                              SendMessage(hEdit, WM_PASTE, 0, 0);
                              // определить текущие координаты курсора
                              SendMessage(hEdit, WM_POSITION, 0, 0);
                              break;
                        case ID_CUT:
                              // вырезать выделенный фрагмент текста
                              SendMessage(hEdit, WM_CUT, 0, 0);
                              // определить текущие координаты курсора
                              SendMessage(hEdit, WM_POSITION, 0, 0);
                              break;
                        case ID_DELETE:
                              // удалить выделенный фрагмент текста
                              SendMessage(hEdit, WM_CLEAR, 0, 0);
                              // определить текущие координаты курсора
                              SendMessage(hEdit, WM_POSITION, 0, 0);
                              break;
                        case ID_SELECTALL:
                              // выделить весь текст в Edit Control
                              SendMessage(hEdit, EM_SETSEL, 0, -1);
                              // определить текущие координаты курсора
                              SendMessage(hEdit, WM_POSITION, 0, 0);
                              break;
                  }
                  return TRUE;
      }
      return FALSE;
}
```

Анализируя вышеприведенный код, следует обратить внимание на переопределение стандартной оконной процедуры текстового поля ввода посредством функции **SetWindowLong**, рассмотренной ранее. Отметим, что в контексте решаемой задачи это достаточно удобно, а в некоторых ситуациях без переопределения оригинальной оконной процедуры элемента управления не обойтись. В конкретном примере переопределённая оконная процедура обрабатывает сообщения мыши, клавиатурные сообщения, а также пользовательское сообщение **WM_POSITION**, т.е. сообщение, определяемое программистом в приложении.

```
#define WM_POSITION WM_APP
```

Как известно, любое системное сообщение – это целочисленное значение. Допускается создавать в приложении пользовательское сообщение. При этом важно, чтобы номер этого сообщения не совпадал с номером системного сообщения. В этой связи неплохо бы знать, для каких целей используется тот или иной целочисленный диапазон.

Рассмотрим следующие диапазоны целочисленных значений:

- диапазон 0 – 0x03FF зарезервирован для системных сообщений (например, #define WM_COMMAND 0x0111);
- диапазон 0x0400 – 0x7FFF обычно используется для идентификаторов окон, создаваемых программно (#define WM_USER 0x0400);
- диапазон 0x8000 – 0xBFFF применяется для определения пользовательских сообщений (#define WM_APP 0x8000);
- диапазон 0xC000 – 0xFFFF предназначен для строковых сообщений (такие сообщения используются при работе с немодальными диалогами и их необходимо регистрировать с помощью функции API **RegisterWindowMessage** – об этом речь пойдёт в последующих уроках);
- целочисленные значения, превышающие 0xFFFF, зарезервированы системой.

Как следует из вышеприведенного кода, переопределённая оконная процедура текстового поля ввода должна вернуть значение, возвращаемое функцией API **CallWindowProc**. Данная функция передаёт все сообщения на обработку стандартной оконной процедуре:

```
LRESULT CallWindowProc(
    WNDPROC lpPrevWndFunc, // указатель на предыдущую (оригинальную) оконную
      // процедуру. Это значение возвращается функцией GetWindowLong.
    HWND hWnd, // дескриптор окна, получающего сообщение
    UINT Msg, // идентификатор сообщения
    WPARAM wParam, // дополнительная информация о сообщении
    LPARAM lParam // дополнительная информация о сообщении
);
```

Кроме того, в приложении используется функция API **IsClipboardFormatAvailable**, которая определяет, имеются ли в буфере обмена данные в указанном формате (например, текстовые данные).

```
BOOL IsClipboardFormatAvailable(
          UINT format // формат данных
     );
```

4. Распаковщики сообщений

Распаковщики сообщений (**Message crackers**) – это специальные макросы, которые упрощают написание оконной процедуры.

Обычно в теле оконной процедуры находится один огромный оператор **switch**, содержащий большое число строк кода, что является образцом плохого стиля программирования. Распаковщики сообщений позволяют разбить оператор **switch** на небольшие функции – по одной на оконное сообщение. Это значительно улучшает внутреннюю структуру кода.

Как известно, с каждым сообщением в оконную процедуру приходит дополнительная информация о сообщении. Эта информация упакована в параметрах **WPARAM** и **LPARAM** и специфична для каждого сообщения. При написании приложений необходимо помнить эту дополнительную информацию или искать её в справочнике. Однако использование макросов упрощает разработку приложений, так как макросы распаковывают параметры сообщений.

Например, для обработки сообщения **WM_CLOSE** необходимо в операторе **switch** оконной процедуры указать макрос **HANDLE_MSG**:

```
BOOL CALLBACK DlgProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
	switch(message)
	{
		HANDLE_MSG(hwnd, WM_CLOSE, Cls_OnClose);
	}
	return FALSE;
}
```

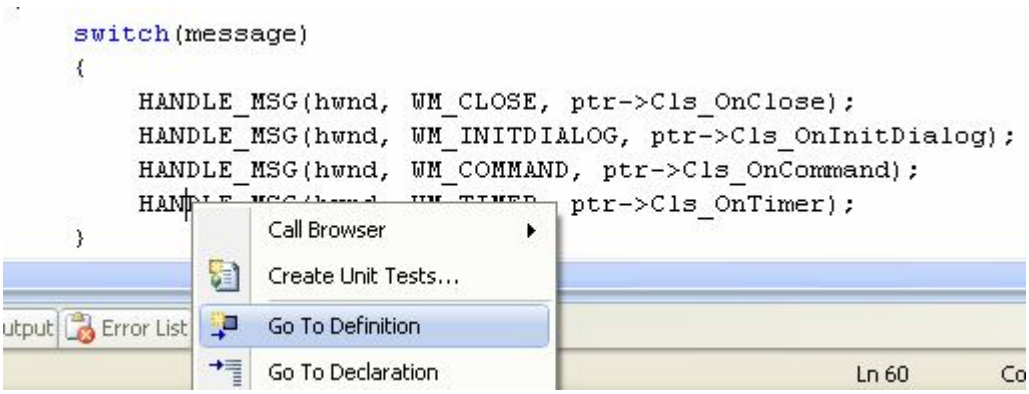
В коде приложения предусмотреть функцию – обработчик сообщения **WM_CLOSE**:

```
void Cls_OnClose(HWND hwnd)
{
	EndDialog(hwnd, 0);
}
```

В файле **WindowsX.h** макрос **HANDLE_MSG** определён так:

```
#define HANDLE_MSG(hwnd, message, fn)    \
    case (message): return HANDLE_##message((hwnd),
       (wParam), (lParam), (fn));
```

Стоит отметить, что для быстрого перехода к месту определения макроса **HANDLE_MSG** в файле **WindowsX.h** необходимо в коде приложения щелкнуть правой кнопкой мыши по макросу, и в появившемся контекстном меню выбрать **Go To Definition**.



Для сообщения **WM_CLOSE** вышеприведенное макроопределение после обработки препроцессором выглядит как:

```
    case (WM_CLOSE):
    return HANDLE_WM_CLOSE((hwnd), (wParam), (lParam), (Cls_OnClose));
```

Макросы **HANDLE_##message** (например, **HANDLE_WM_CLOSE**, **HANDLE_WM_COMMAND** и т.д.) представляют собой распаковщики сообщений. Они распаковывают содержимое параметров **wParam** и **lParam**, выполняют нужные преобразования типов и вызывают соответствующую функцию – обработчик сообщения (например, **Cls_OnClose**). Например, макрос **HANDLE_WM_CLOSE** в файле **WindowsX.h** определён следующим образом:

```
    #define HANDLE_WM_CLOSE(hwnd, wParam, lParam, fn) ((fn)(hwnd), 0L)
```

Результат раскрытия препроцессором этого макроса – вызов функции **Cls_OnClose**, которой передаются распакованные части параметров **wParam** и **lParam**. При этом производятся соответствующие преобразования типов.

Чтобы использовать распаковщик для обработки сообщения, например, **WM_COMMAND**, следует найти в файле **WindowsX.h** следующий фрагмент кода:

```
/* void Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) */
#define HANDLE_WM_COMMAND(hwnd, wParam, lParam, fn) \
            ((fn)((hwnd), (int)(LOWORD(wParam)), (HWND)(lParam),
            (UINT)HIWORD(wParam)), 0L)

#define FORWARD_WM_COMMAND(hwnd, id, hwndCtl, codeNotify, fn) \
          (void)(fn)((hwnd), WM_COMMAND,
          MAKEWPARAM((UINT)(id),(UINT)(codeNotify)), (LPARAM)(HWND)(hwndCtl))
```

Первая строка в этом коде - прототип функции – обработчика сообщения **WM_COMMAND**. Следующая строка – распаковщик сообщения. Последняя строка в этом фрагменте кода содержит предописатель

сообщения (**message forwarder**), который используется в том случае, когда при обработке сообщения необходимо вызвать стандартный обработчик по умолчанию.

```
void Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{
	// выполняем обработку сообщения


	// обработка по умолчанию
	FORWARD_WM_COMMAND(hwnd, id, hwndCtl, codeNotify, DefWindowProc);
}
```

Для демонстрации использования распаковщиков сообщений модифицируем код приложения «Слайд-шоу», рассмотренного ранее (исходный код приложения находится в папке **SOURCE / Message crackers**).

```
// header.h

#pragma once

#include <windows.h>
#include <WindowsX.h>
#include "resource.h"


// CMessageCrackersDlg.h


#pragma once
#include "header.h"

class CMessageCrackersDlg
{
public:
	CMessageCrackersDlg(void);


public:
	static BOOL CALLBACK DlgProc(HWND hWnd, UINT mes, WPARAM wp, LPARAM lp);
	static CMessageCrackersDlg *ptr;
	BOOL Cls_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);
	void Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify);
	void Cls_OnClose(HWND hwnd);
	void Cls_OnTimer(HWND hwnd, UINT id);
	HWND hDialog;
	HWND hStart, hStop, hPicture;
	HBITMAP hBmp[5];
};
```

```
                        // CMessageCrackersDlg.cpp


#include "CMessageCrackersDlg.h"

CMessageCrackersDlg* CMessageCrackersDlg::ptr = NULL;

CMessageCrackersDlg::CMessageCrackersDlg(void)
{
      ptr = this;
}

void CMessageCrackersDlg::Cls_OnClose(HWND hwnd)
{
      EndDialog(hwnd, 0);
}

BOOL CMessageCrackersDlg::Cls_OnInitDialog(HWND hwnd, HWND hwndFocus,
                                           LPARAM lParam)
{
      hDialog = hwnd;
      hStart = GetDlgItem(hDialog, IDC_START);
      hStop = GetDlgItem(hDialog, IDC_STOP);
      hPicture = GetDlgItem(hDialog, IDC_PICTURE);
      for(int i = 0; i < 5; i++)
            hBmp[i] = LoadBitmap(GetModuleHandle(0),
                        MAKEINTRESOURCE(IDB_BITMAP1 + i));
      return TRUE;
}

void CMessageCrackersDlg::Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl,
                                           UINT codeNotify)
{
      if(id == IDC_START)
      {
            SetTimer(hDialog, 1, 1000, 0);
            EnableWindow(hStart, 0);
            EnableWindow(hStop, 1);
            SetFocus(hStop);
      }
      else if(id == IDC_STOP)
      {
            KillTimer(hDialog, 1);
            EnableWindow(hStart, 1);
            EnableWindow(hStop, 0);
            SetFocus(hStart);
      }
}

void CMessageCrackersDlg::Cls_OnTimer(HWND hwnd, UINT id)
{
      static int index = 0;
      index++;
      if(index > 4)
            index = 0;
      SendMessage(hPicture, STM_SETIMAGE, WPARAM(IMAGE_BITMAP),
                  LPARAM(hBmp[index]));
}
```

```
BOOL CALLBACK CMessageCrackersDlg::DlgProc(HWND hwnd, UINT message,
                                    WPARAM wParam, LPARAM lParam)
{

      switch(message)
      {
            HANDLE_MSG(hwnd, WM_CLOSE, ptr->Cls_OnClose);
            HANDLE_MSG(hwnd, WM_INITDIALOG, ptr->Cls_OnInitDialog);
            HANDLE_MSG(hwnd, WM_COMMAND, ptr->Cls_OnCommand);
            HANDLE_MSG(hwnd, WM_TIMER, ptr->Cls_OnTimer);
      }
      return FALSE;
}


                           // CMessageCrackers.cpp


#include "CMessageCrackersDlg.h"


int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrev, LPSTR lpszCmdLine,
                   int nCmdShow)
{
      CMessageCrackersDlg dlg;
      return DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                        CMessageCrackersDlg::DlgProc);
}
```

В завершении рассмотрения данного вопроса ещё раз отметим преимущества распаковщиков сообщений:

- сокращение числа явных преобразований типов в коде приложения, а также возникающих при этом ошибок;
- читабельность кода;
- простота и удобство в использовании при разработке приложения.

Домашнее задание

1. Написать приложение «Простейший калькулятор» на четыре действия (сложение, вычитание, умножение и деление). На форме диалога расположены три текстовых поля для ввода операндов и знака операции, кнопка, при нажатии на которую, подсчитывается результат, а также статик для вывода результата вычисления.
2. Написать приложение, которое по введенной дате определяет день недели. При этом день, месяц и год необходимо вводить в отдельные текстовые поля. Результат также следует выводить в текстовое поле со стилем Read Only. Предусмотреть проверку корректности ввода даты.
3. Написать приложение, определяющее, сколько осталось времени до указанной даты (день, месяц и год вводятся в отдельные текстовые поля). Предусмотреть выдачу результата в годах, месяцах, днях.
4. Написать игру «Пятнашки», учитывая следующие требования:
   - предусмотреть автоматическую перестановку «пятнашек» в начале новой игры;
   - выводить время, за которое пользователь окончил игру (собрал «пятнашки»);
   - предусмотреть возможность начать новую игру.