Модуль №3

Занятие №1

Версия 1.0.1

План занятия:

1. Повторение пройденного материала.
2. Элемент управления «кнопка».
    2.1. Обычная кнопка (Button).
    2.2. Флажок (Check Box).
    2.3. Переключатель (Radio Button).
3. Практическая часть.
4. Подведение итогов.
5. Домашнее задание.

1. Повторение пройденного материала

Данное занятие необходимо начать с краткого повторения материала предыдущего занятия. При общении со слушателями можно использовать следующие контрольные вопросы:

1) Что такое синхронное сообщение?
2) Что такое асинхронное сообщение?
3) Каким образом можно отправить синхронное сообщение?
4) Каким образом можно отправить асинхронное сообщение?
5) В чём принципиальное отличие функции **SendMessage** от функции **PostMessage**?

2. Элемент управления «кнопка»

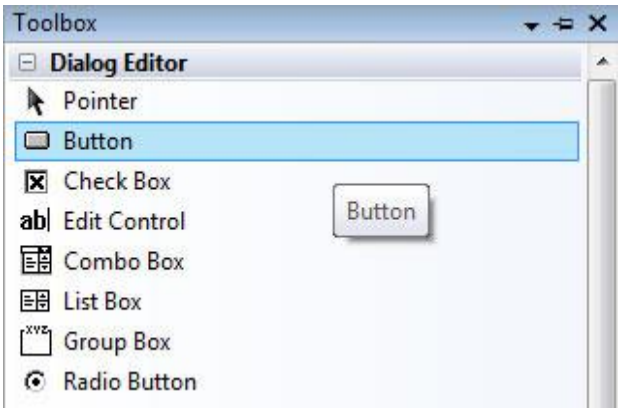
2.1. Обычная кнопка (Button)

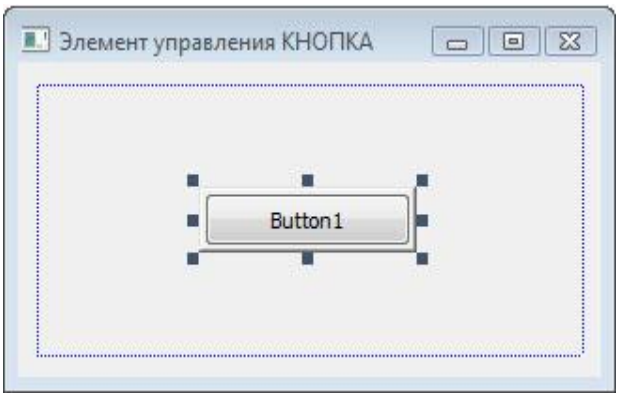
Напомнить слушателям, что они уже знакомы с данным элементом управления при использовании окон сообщений (**MessageBox**).

Создать кнопку на форме диалога можно двумя способами:

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

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



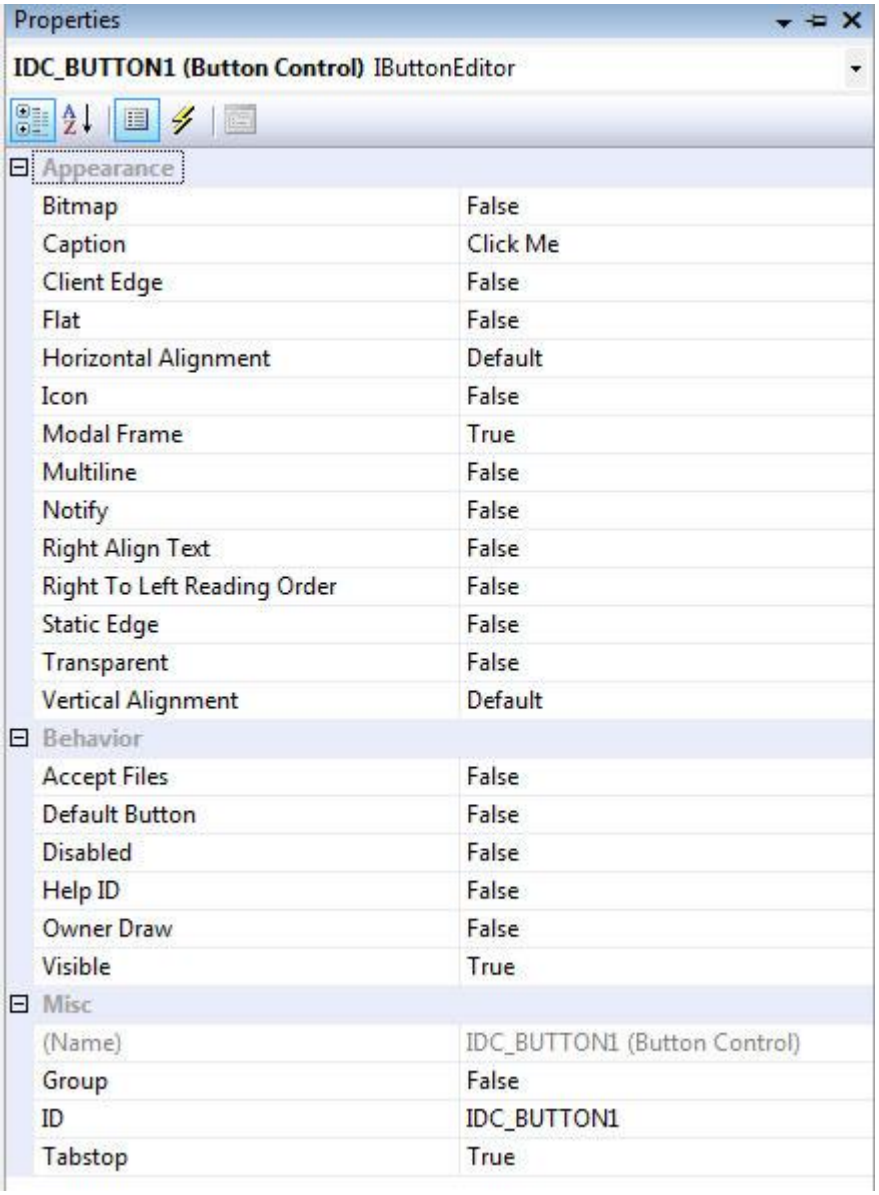


После размещения кнопки на форме диалога ей назначается идентификатор (например, **IDC_BUTTON1**), который впоследствии можно изменить на идентификатор, отражающий семантику ресурса.

Рекомендуется ознакомить слушателей с некоторыми свойствами элемента управления **Button**.

- Свойство **Caption** содержит текстовую строку, которая будет отображаться внутри ограничивающего прямоугольника элемента **Button**. При этом в строке могут использоваться управляющие символы **\t** (табуляция) и **\n** (перевод строки).

- Свойство **Multiline** позволяет располагать текст в несколько строк.
- Свойства **Horizontal Alignment** и **Vertical Alignment** позволяют выбрать вариант выравнивания текста внутри ограничивающего прямоугольника.
- Свойство **Flat** позволяет создать плоскую кнопку.
- Свойство **Default Button** назначает кнопке атрибут «применяемая по умолчанию».



- Свойства **Icon** или **Bitmap** позволяют указать, что вместо текста на кнопке будет отображаться пиктограмма или растровый образ.

- При истинном значении свойства **Notify** кнопка будет отправлять уведомление **BN_CLICKED** родительскому окну (диалогу).

В качестве примера, демонстрирующего использование кнопок, привести следующий код:

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

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

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

int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                     LPTSTR 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:
                  hStart = GetDlgItem(hWnd, IDC_START);
                  hStop = GetDlgItem(hWnd, IDC_STOP);
                  hPicture = GetDlgItem(hWnd, IDC_PICTURE);
                  for(int i = 0; i < 5; i++)
                        hBmp[i] = LoadBitmap(GetModuleHandle(NULL),
                                    MAKEINTRESOURCE(IDB_BITMAP1 + i));
                  return TRUE;

            case WM_COMMAND:
                  if(LOWORD(wParam) == IDC_START)
                  {
                        SetTimer(hWnd, 1, 1000, NULL);
                        EnableWindow(hStart, FALSE);
                        EnableWindow(hStop, TRUE);
                        SetFocus(hStop);
                  }
                  else if(LOWORD(wParam) == IDC_STOP)
                  {
                        KillTimer(hWnd, 1);
                        EnableWindow(hStart, TRUE);
                        EnableWindow(hStop, FALSE);
                        SetFocus(hStart);
                  }
                  return TRUE;
```

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

Анализируя вышеприведенный код, следует отметить, что при воздействии на элемент управления диалога (в данном случае, при нажатии на кнопку) в диалоговую процедуру **DlgProc** поступает сообщение **WM_COMMAND,** в котором **LOWORD(wParam)** содержит идентификатор элемента управления, **HIWORD(wParam)** содержит код уведомления (в данном случае, **BN_CLICKED)**, а **lParam** – дескриптор элемента управления.

Следует подчеркнуть, если кнопка имеет фокус ввода, то текст на кнопке обводится штриховой линией, а нажатие и отпускание клавиши пробела имеет тот же эффект, что и щелчок мышью по кнопке. Существует программный способ перевода фокуса ввода на элемент управления. Для этой цели служит функция API **SetFocus**:

```
HWND SetFocus(
HWND hWnd // дескриптор окна, приобретающего клавиатурный ввод
);
```

Для получения дескриптора окна (элемента управления), обладающего фокусом ввода используется функция API **GetFocus**:

```
HWND GetFocus(VOID);
```

Альтернативный способ создания кнопки - использование функции **CreateWindowEx**, рассмотренной на одном из предыдущих занятий. В этом случае

во втором параметре функции передается имя предопределенного оконного класса – **BUTTON**.

В качестве примера, демонстрирующего программный способ создания кнопок, привести следующий код:

```
#include <windows.h>
#include <tchar.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 _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                     LPTSTR 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, NULL);
                            EnableWindow(hStart, FALSE);
                            EnableWindow(hStop, TRUE);
                            SetFocus(hStop);
                      }
                      else if(!lstrcmp(text, TEXT("Stop")))
                      {
                            KillTimer(hWnd, 1);
                            EnableWindow(hStart, TRUE);
                            EnableWindow(hStop, FALSE);
                            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, акцентировать внимание слушателей на стилях, использованных 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. Напомнить слушателям, что запрещённые элементы выводятся на экран серым цветом и не воспринимают пользовательский ввод с клавиатуры или от мыши.

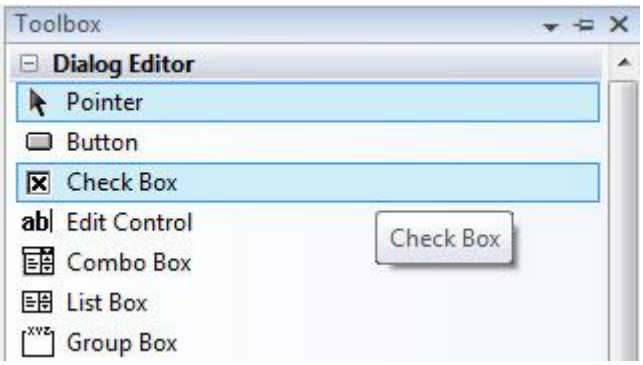
2.2. Флажок (Check Box)

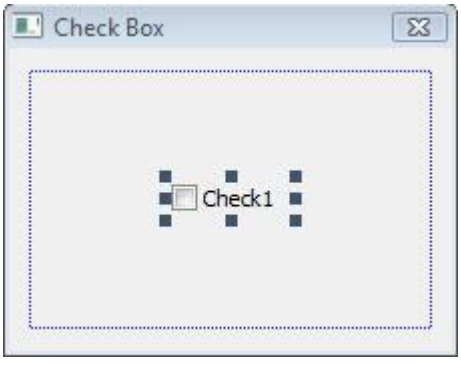
Данный элемент управления обычно применяется для установки или сброса определённых опций, независимых друг от друга. Флажок действует как двухпозиционный переключатель. Один щелчок вызывает появление контрольной отметки (галочки), а другой щелчок приводит к ее исчезновению.

Создать **Check Box** на форме диалога можно двумя способами:

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

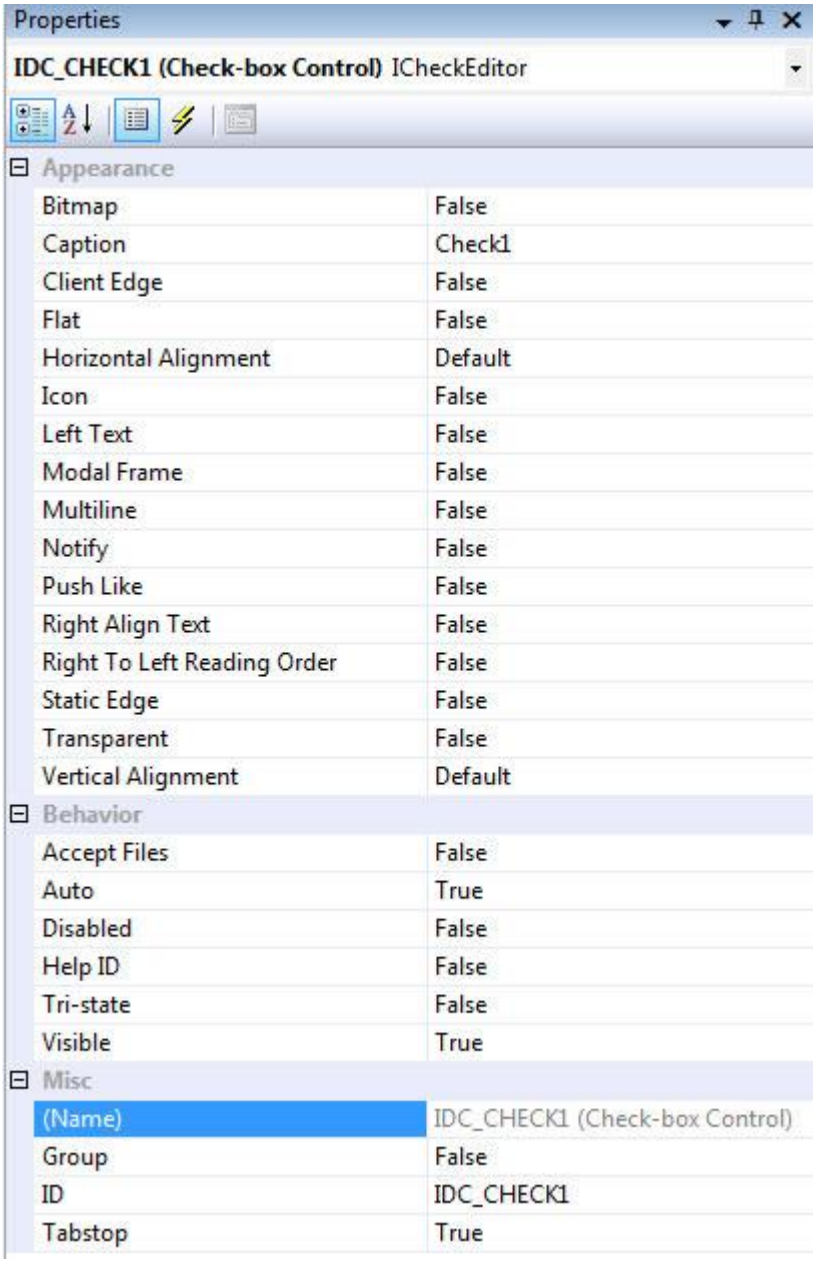
При первом способе необходимо определить **Check Box** в шаблоне диалогового окна на языке описания шаблона диалога. Это произойдёт автоматически, если активизировать окно **Toolbox** (<**Ctrl**><**Alt**><**X**>) и «перетащить» **Check Box** на форму диалога.





После размещения флажка на форме диалога ему назначается идентификатор (например, **IDC_CHECK1**), который впоследствии можно изменить на идентификатор, отражающий семантику ресурса.

Следует отметить, что элемент управления **Check Box** обладает тем же набором свойств, что и **Button**, а также располагает дополнительными свойствами, характерными только для него.



- Свойство **Auto** позволяет элементу управления отслеживать все щелчки мышью, и при этом элемент управления сам включает или выключает контрольную отметку. Если же отключить свойство **Auto**, то управление флажком полностью возлагается на приложение.

- Свойство **Tri-state** используется для создания флажка, имеющего три состояния. Кроме состояний «установлен» и «сброшен» добавляется «неопределенное состояние», в котором флажок отображен в серой гамме. Серый цвет показывает пользователю, что выбор флажка не определен или не имеет отношения к текущей операции.
- Свойство **Push-like** изменяет внешний вид флажка так, что он выглядит как нажимаемая кнопка. Вместо установки галочки эта кнопка переходит в нажатое состояние и остается в нем до следующего щелчка мышью.

В качестве примера, демонстрирующего использование элемента управления **Check Box**, привести следующий код:

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

HWND hStart, hDateTime, hShowSeconds;

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

int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                     LPTSTR 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:
                  hStart = GetDlgItem(hWnd, IDC_START);
                  hDateTime = GetDlgItem(hWnd, IDC_DATE_TIME);
                  hShowSeconds = GetDlgItem(hWnd, IDC_SHOW_SECONDS);
                  SendMessage(hShowSeconds, BM_SETCHECK,
                              WPARAM(BST_CHECKED), 0);
                  return TRUE;
            case WM_COMMAND:
                  if(LOWORD(wParam) == IDC_START)
                  {
                        TCHAR str[10];
                        GetWindowText(hStart, str, 10);
                        if(!lstrcmp(str, TEXT("Start")))
                        {
```

```
                        SetTimer(hWnd, 1, 1000, NULL);
                        SetWindowText(hStart, TEXT("Stop"));
                    }
                    else if(!lstrcmp(str, TEXT("Stop")))
                    {
                        KillTimer(hWnd, 1);
                        SetWindowText(hStart, TEXT("Start"));
                        SetWindowText(hDateTime, NULL);
                    }
                }
                return TRUE;
            case WM_TIMER:
                {
                    static time_t t;
                    static TCHAR str[100];
                    t = time(NULL);
                    struct tm DateTime= *(localtime(&t));
                    LRESULT lResult = SendMessage(hShowSeconds,
                                      BM_GETCHECK, 0, 0);
                    if(lResult == BST_CHECKED)
                        _tcsftime(str, 100,
                        TEXT("%H:%M:%S  %d.%m.%Y  %A"), &DateTime);
                    else
                        _tcsftime(str, 100,
                        TEXT("%H:%M  %d.%m.%Y  %A"), &DateTime);
                    SetWindowText(hDateTime, str);
                }
                return TRUE;
        }
        return FALSE;
}
```

Анализируя вышеприведенный код, следует отметить, для того, чтобы перевести **Check Box** в некоторое состояние, ему необходимо отправить сообщение **BM_SETCHECK**, передав в **WPARAM** одно из следующих значений:

- **BST_CHECKED** - установить отметку;
- **BST_UNCHECKED** – снять отметку;
- **BST_INDETERMINATE** - установить неопределенное состояние.

Существует альтернативный способ программной инициализации состояния элемента управления **Check Box**. Для этого используется функция API **CheckDlgButton**:

```
BOOL CheckDlgButton(
    HWND hDlg, // дескриптор диалога, содержащего кнопку (флажок)
    int nIDButton, // идентификатор элемента управления (флажка)
    UINT uCheck // состояние флажка - одно из вышеперечисленных значений
```

```
);
```

Для получения состояния флажка следует ему послать сообщение **BM_GETCHECK**. В этом случае **SendMessage** вернёт одно из вышеперечисленных значений.

Альтернативным способом получения состояния флажка является вызов функции API **IsDlgButtonChecked**:

```
UINT IsDlgButtonChecked(
     HWND hDlg, // дескриптор диалога, содержащего кнопку (флажок)
     int nIDButton // идентификатор элемента управления (флажка)
);
```

В качестве примера, демонстрирующего программный способ создания флажка, привести следующий фрагмент кода:

```
HWND h = CreateWindowEx(0, TEXT("BUTTON"),TEXT("CheckBox"),
            WS_CHILD | WS_VISIBLE | BS_AUTOCHECKBOX,
              LEFT, TOP, WIDTH, HEIGHT, hWnd, 0, GetModuleHandle(NULL), 0);
```

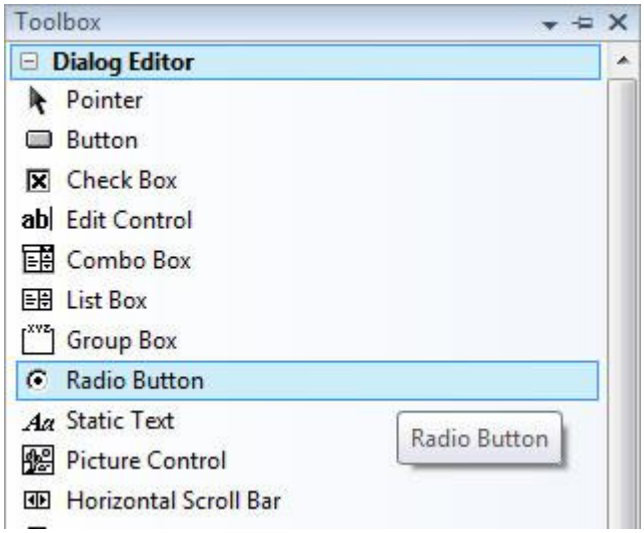
2.3. Переключатель (Radio Button)

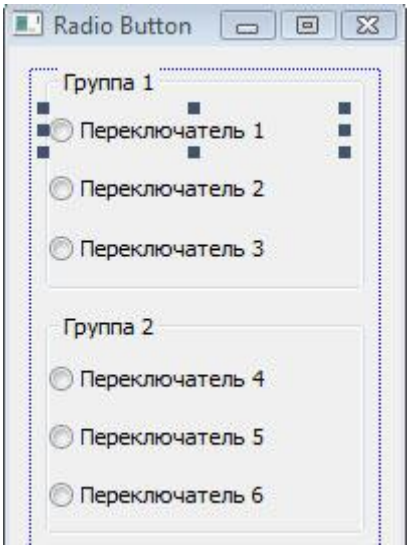
Данный элемент управления обычно применяется для представления в окне множества взаимоисключающих опций, из которых можно выбрать только одну. В отличие от флажков, повторный щелчок на переключателе не меняет его состояние.

Создать **Radio Button** на форме диалога можно двумя способами:

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

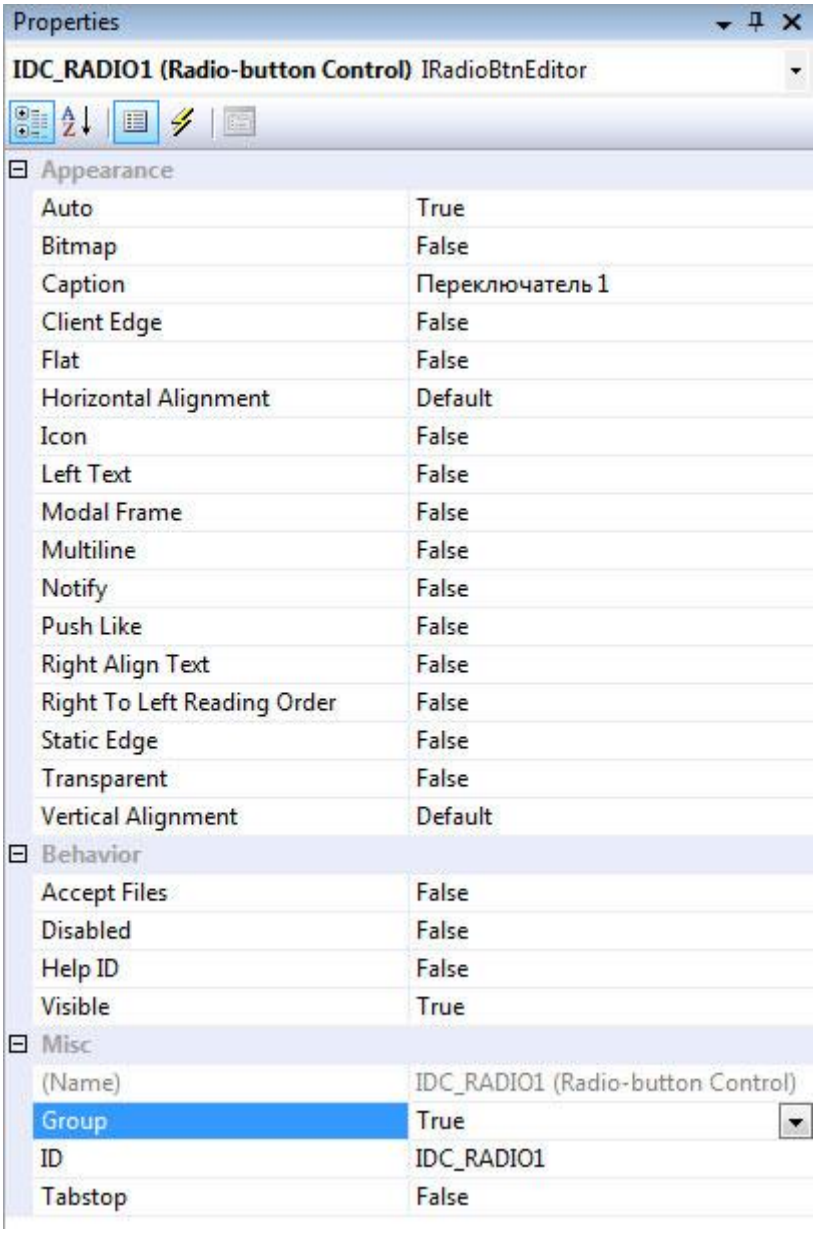
При первом способе необходимо определить **Radio Button** в шаблоне диалогового окна на языке описания шаблона диалога. Это произойдёт автоматически, если активизировать окно **Toolbox** (<**Ctrl**><**Alt**><**X**>) и «перетащить» **Radio Button** на форму диалога.





После размещения переключателя на форме диалога ему назначается идентификатор (например, **IDC_RADIO1**), который впоследствии можно изменить на идентификатор, отражающий семантику ресурса.

Следует отметить, что элемент управления **Radio Button** обладает тем же набором свойств, что и **Check Box**, но, кроме того, имеет важное свойство **Group**. Для первого переключателя в группе связанных взаимоисключающих переключателей нужно обязательно установить значение свойства **Group**, равным **True**. Все последующие переключатели (в файле описания ресурсов) со значением свойства **Group**, равным **False**, считаются принадлежащими к этой группе. Если в последовательности описаний элементов управления встречается переключатель со значением свойства **Group**, равным **True**, считается, что он начинает новую группу элементов **Radio Button**.



В качестве примера, демонстрирующего использование элемента управления **Radio Button**, привести следующий код:

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

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

int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                 LPTSTR lpszCmdLine, int nCmdShow)
{
      return DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                       DlgProc);
}
```

```
BOOL CALLBACK DlgProc(HWND hWnd, UINT message, WPARAM wp, LPARAM lp)
{

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

		case  WM_INITDIALOG:
			SendDlgItemMessage(hWnd, IDC_RADIO2, BM_SETCHECK,
					WPARAM(BST_CHECKED), 0);
			SendDlgItemMessage(hWnd, IDC_RADIO5, BM_SETCHECK,
					WPARAM(BST_CHECKED), 0);
			return TRUE;

		case WM_COMMAND:
			if(LOWORD(wp) >= IDC_RADIO1 && LOWORD(wp) <= IDC_RADIO6)
			{
				TCHAR str[20];
				wsprintf(str, TEXT("Переключатель %d"),
					LOWORD(wp) - IDC RADIO1 + 1);
				SetWindowText(hWnd, str);
			}
			else if(LOWORD(wp)==IDC_BUTTON1)
			{
				TCHAR str[100] =
					TEXT("Выбраны следующие переключатели:\n");

				// Анализируем первую группу переключателей
				LRESULT result = SendDlgItemMessage(hWnd, IDC_RADIO1,
						BM_GETCHECK, 0, 0);
				if(result == BST_CHECKED)
					lstrcat(str, TEXT("Радио №1\n"));
				else
				{
					result = SendDlgItemMessage(hWnd, IDC_RADIO2,
							BM_GETCHECK, 0, 0);
					if(result == BST_CHECKED)
						lstrcat(str, TEXT("Радио №2\n"));
					else
					{
						result = SendDlgItemMessage(hWnd,
							IDC_RADIO3, BM_GETCHECK, 0, 0);
						if(result == BST_CHECKED)
							lstrcat(str, TEXT("Радио №3\n"));
					}
				}

				// Анализируем вторую группу переключателей
				result = SendDlgItemMessage(hWnd, IDC_RADIO4,
								BM_GETCHECK, 0, 0);
				if(result == BST_CHECKED)
					lstrcat(str, TEXT("Радио №4\n"));
				else
				{
					result = SendDlgItemMessage(hWnd, IDC_RADIO5,
								BM_GETCHECK, 0, 0);
```

```
                        if(result == BST_CHECKED)
                              lstrcat(str, TEXT("Радио №5\n"));
                        else
                        {
                              result = SendDlgItemMessage(hWnd,
                                    IDC_RADIO6, BM_GETCHECK, 0, 0);
                              if(result == BST_CHECKED)
                                    lstrcat(str, TEXT("Радио №6\n"));
                        }
                  }

                  MessageBox(hWnd, str, TEXT("Radio Button"),
                              MB_OK | MB_ICONINFORMATION);
            }
            return TRUE;
      }
      return FALSE;
}
```

Анализируя вышеприведенный код, следует отметить, для того, чтобы перевести **Radio Button** в некоторое состояние, ему необходимо отправить сообщение **BM_SETCHECK**, передав в **WPARAM** одно из следующих значений:

- **BST_CHECKED** - установить отметку;
- **BST_UNCHECKED** – снять отметку;
- **BST_INDETERMINATE** - установить неопределенное состояние.

Существует альтернативный способ выбора переключателя. Для этого используется функция API **CheckRadioButton**:

```
BOOL CheckRadioButton(
    HWND hDlg, // дескриптор диалога, содержащего кнопку (переключатель)
    int nIDFirstButton, // идентификатор первого переключателя в группе
    int nIDLastButton, // идентификатор последнего переключателя в группе
    int nIDCheckButton // идентификатор выбираемого переключателя
);
```

Данная функция помечает указанный переключатель в группе, удаляя отметку со всех других переключателей этой же группы. Другими словами, функция **CheckRadioButton** посылает сообщение **BM_SETCHECK** каждому переключателю указанной группы. При этом выбираемому переключателю в **WPARAM** передается **BST_CHECKED,** а остальным **- BST_UNCHECKED.**

Для получения состояния переключателя следует ему послать сообщение **BM_GETCHECK**. В этом случае **SendMessage** (либо **SendDlgItemMessage**) вернёт одно из вышеперечисленных значений.

Альтернативным способом получения состояния переключателя является вызов функции API **IsDlgButtonChecked**, рассмотренной ранее.

Акцентировать внимание слушателей на применении в вышеприведенном коде функции API **SendDlgItemMessage**, предназначенной для посылки сообщений элементам управления.

```
LRESULT SendDlgItemMessage(
    HWND hDlg, // дескриптор диалога, содержащего элемент управления
    int nIDDlgItem, // идентификатор дочернего окна (элемента управления),
                    // которому отправляется сообщение
    UINT Msg, // идентификатор сообщения
    WPARAM wParam, // дополнительная информация о сообщении
    LPARAM lParam  // дополнительная информация о сообщении
);
```

В качестве примера, демонстрирующего программный способ создания переключателей, привести следующий фрагмент кода:

```
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);
}
```

3. Практическая часть

Написать игру «Крестики-нолики», учитывая следующие требования:

- игровое поле размером 3x3 должно состоять из кнопок;
- при нажатии на кнопку, на ней должна отобразиться картинка (крестик или нолик);

- необходимо предотвращать попытку поставить крестик или нолик на занятую клетку;
- предоставить пользователю право выбора первого хода, используя флажок;
- предусмотреть возможность выбора уровня сложности, используя переключатели;
- предусмотреть кнопку «Начать новую игру».

4. Подведение итогов

Подвести общие итоги занятия. Ещё раз отметить, для каких целей используется обычная кнопка, флажок и переключатель. Выделить два основных способа создания кнопок – с помощью средств интегрированной среды разработки приложений, а также программный способ. Акцентировать внимание слушателей на наиболее тонких моментах изложенной темы.

5. Домашнее задание

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.

Copyright © 2010 Виталий Полянский