Модуль №2

Занятие №1

Версия 1.0.1

План занятия:

1. Повторение пройденного материала.
2. Диалоговое приложение.
   2.1. Модальный диалог.
   2.2. Немодальный диалог.
3. Создание диалогового приложения.
   3.1. Создание приложения на основе модального диалога.
   3.2. Создание приложения на основе немодального диалога.
4. Общие сведения об элементах управления.
   4.1. Статический элемент управления **Static Text**.
   4.2. Статический элемент управления **Picture Control**.
5. Практическая часть.
6. Подведение итогов.
7. 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) Каким образом в приложении можно установить таймер?
2) Какие существуют способы обработки прерываний таймера?
3) Каким образом таймер можно остановить?

4) С помощью какой функции можно определить, какие приложения, владеющие окном, выполняются в данное время?
5) Каков принцип работы функции, перечисляющей окна верхнего уровня?
6) Какая функция может применяться для последовательной обработки дочерних окон?
7) Каков принцип работы функции, перечисляющей дочерние окна?
8) Какие объекты определяются в ресурсах приложения? Привести примеры.
9) Что такое файл описания ресурсов (**resource script**)?
10) Какая информация хранится в заголовочном файле **resource.h**?
11) Чем отличается загрузка предопределённого ресурса от загрузки нестандартного ресурса, определённого в приложении?
12) Для какой цели используется макрос **MAKEINTRESOURCE?**
13) Какими способами можно получить дескриптор приложения в оконной процедуре?
14) Что такое пиктограмма (иконка)? Какие типовые размеры иконки применяются чаще всего?
15) Что такое курсор? С помощью какого инструмента можно назначить активную точку?
16) Какая функция позволяет модифицировать оконный класс?
17) Какая функция служит для динамического изменения формы курсора в зависимости от его местонахождения?
18) Каким образом можно проконтролировать успешность работы функции API, а также получить описание ошибки при её возникновении в результате работы функции?
19) Каково назначение утилиты **Error Lookup**?

2. Диалоговое приложение

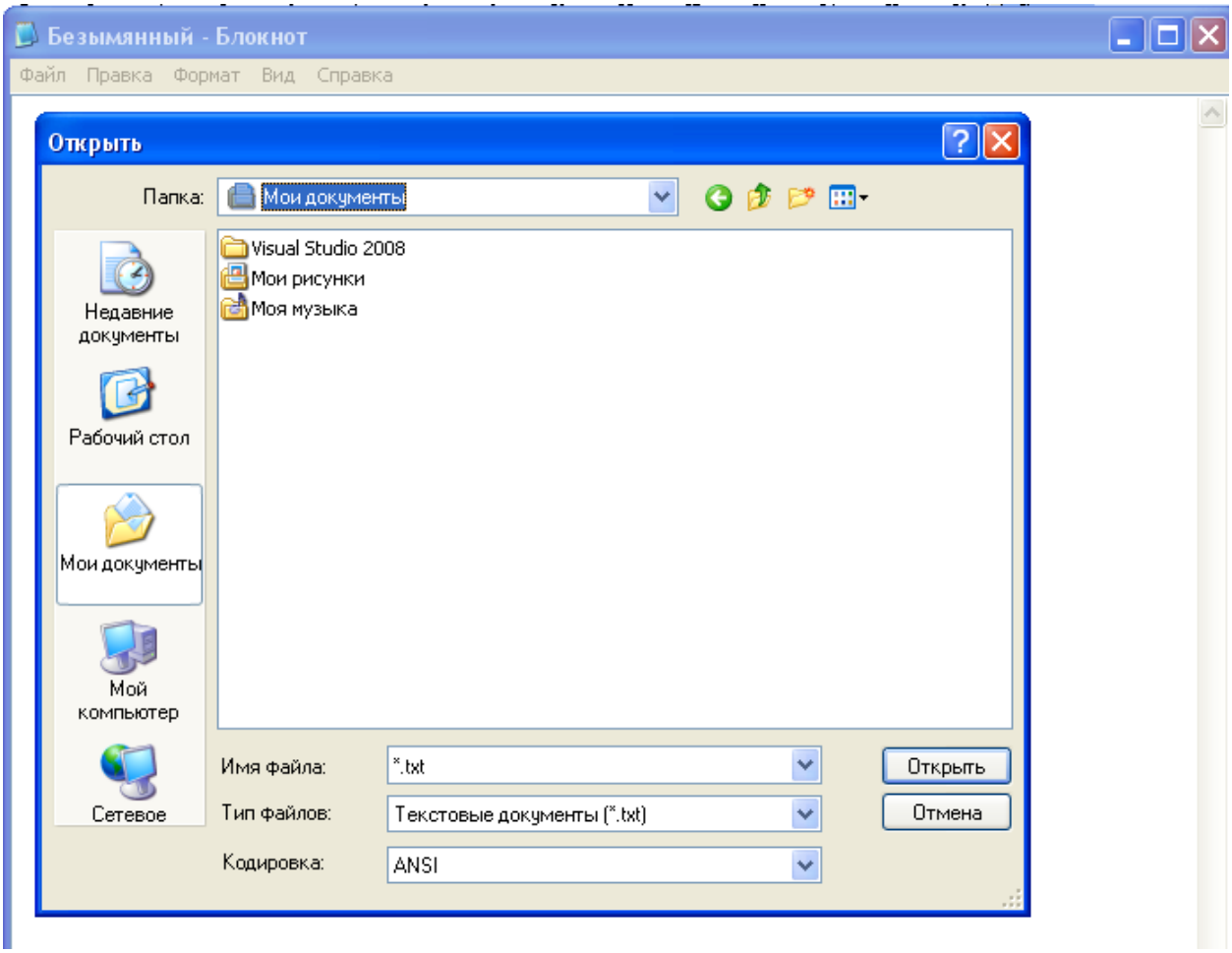
Рассмотрение данного вопроса следует начать с понятий диалога и элемента управления. **Диалог** – это специальный тип окна, предоставляющий интерфейс для взаимодействия пользователя с приложением. Взаимодействие с пользователем осуществляется посредством элементов управления, которые являются дочерними окнами по отношению к диалоговому окну. **Элемент управления** представляет собой особый тип окна, предназначенный для ввода или вывода информации. Примером элемента управления является кнопка, список, текстовое поле, переключатель и ряд других элементов.

Далее следует отметить, что диалоги бывают двух типов: **модальные** (**modal**) и **немодальные (modeless)**. В большинстве случаев используются модальные диалоги.

2.1. Модальный диалог

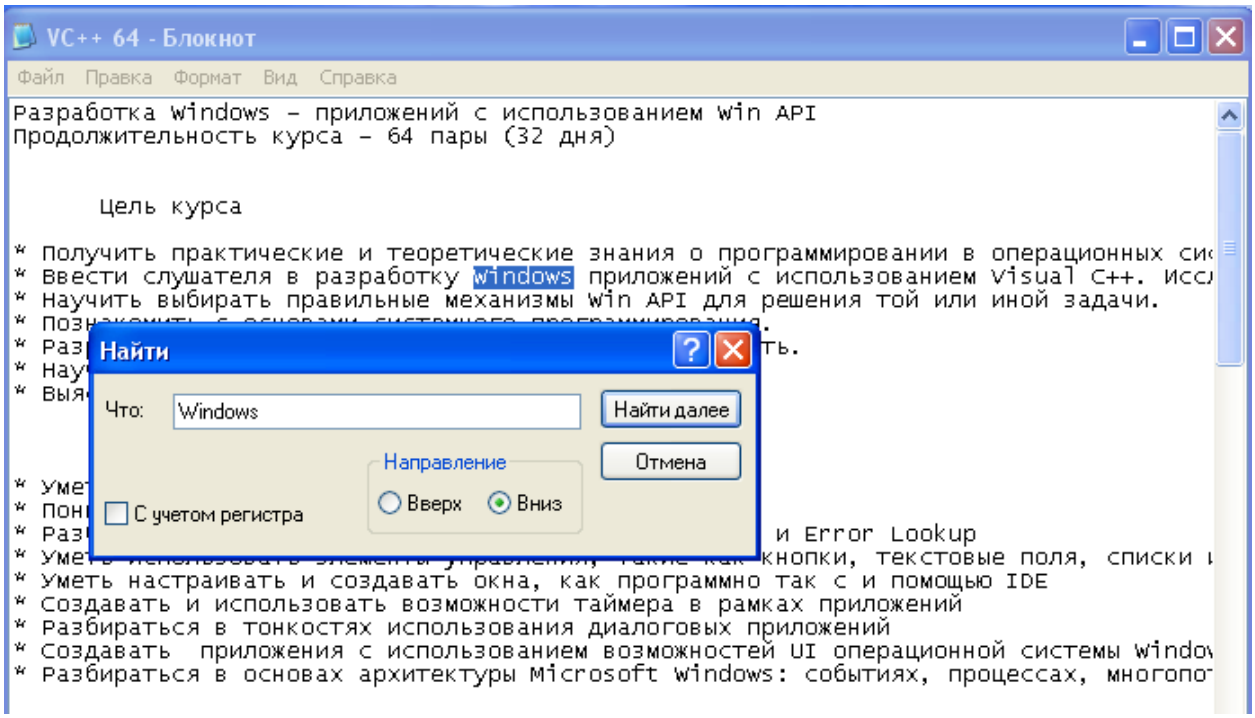
Особенность модального диалога состоит в том, что приложение, создав диалог, всегда дожидается его закрытия, после чего приложение продолжает свою работу. Модальный диалог не позволяет переключить ввод на другие окна, порожденные приложением. При этом пользователь может переключаться в другие программы, не закрыв диалоговое окно. Примером модального диалога может послужить диалоговое окно «Открыть» приложения «Блокнот».

Существует также специальный вид модальных диалоговых окон — **системные модальные (system modal)** окна, которые не позволяют переключаться даже в другие программы. Они сообщают о серьезных проблемах, и пользователь должен закрыть системное модальное окно, чтобы продолжить работу в Windows.



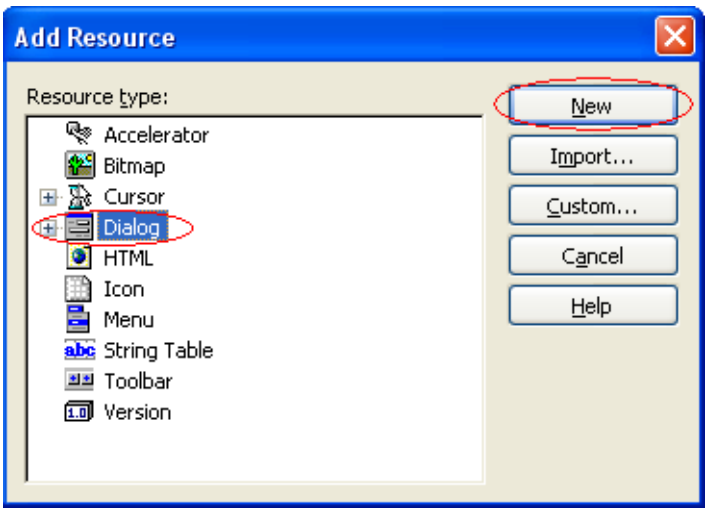
2.2. Немодальный диалог

Немодальный диалог не задерживает выполнение программы, то есть для ее продолжения не требуется завершение диалога. При этом разрешается переключение между диалогом и другими окнами приложения. Таким образом, немодальный диалог может получать и терять фокус ввода. Диалоги этого типа предпочтительней использовать в тех случаях, когда они содержат элементы управления, которые должны быть в любой момент доступны пользователю. Примером немодального диалога может послужить диалоговое окно «Найти» приложения «Блокнот».

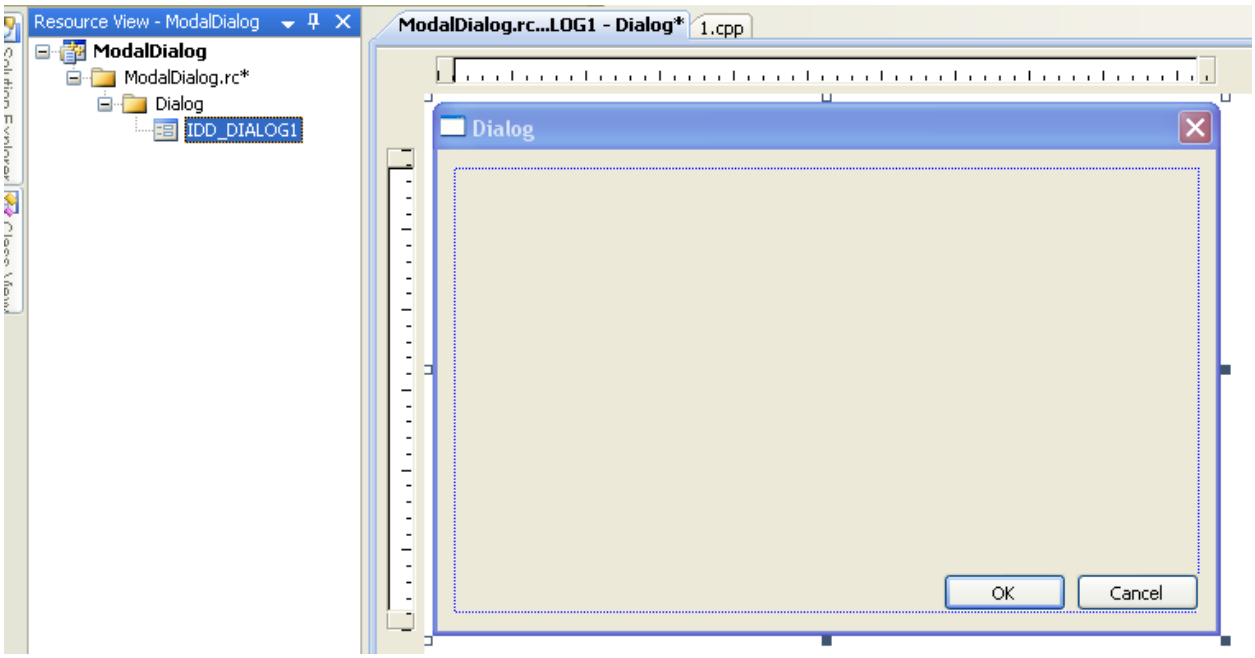


3. Создание диалогового приложения

После изложения общих сведений о диалогах, следует рассмотреть вопрос создания приложения на основе диалога. Для этого, прежде всего, необходимо определить шаблон диалога в файле описания ресурсов. Для этого необходимо активизировать вкладку **Resource View**, в которой с помощью контекстного меню вызвать диалог добавления ресурса **Add -> Resource...**



В этом диалоговом окне необходимо выбрать из списка **Dialog** и нажать кнопку **New**.



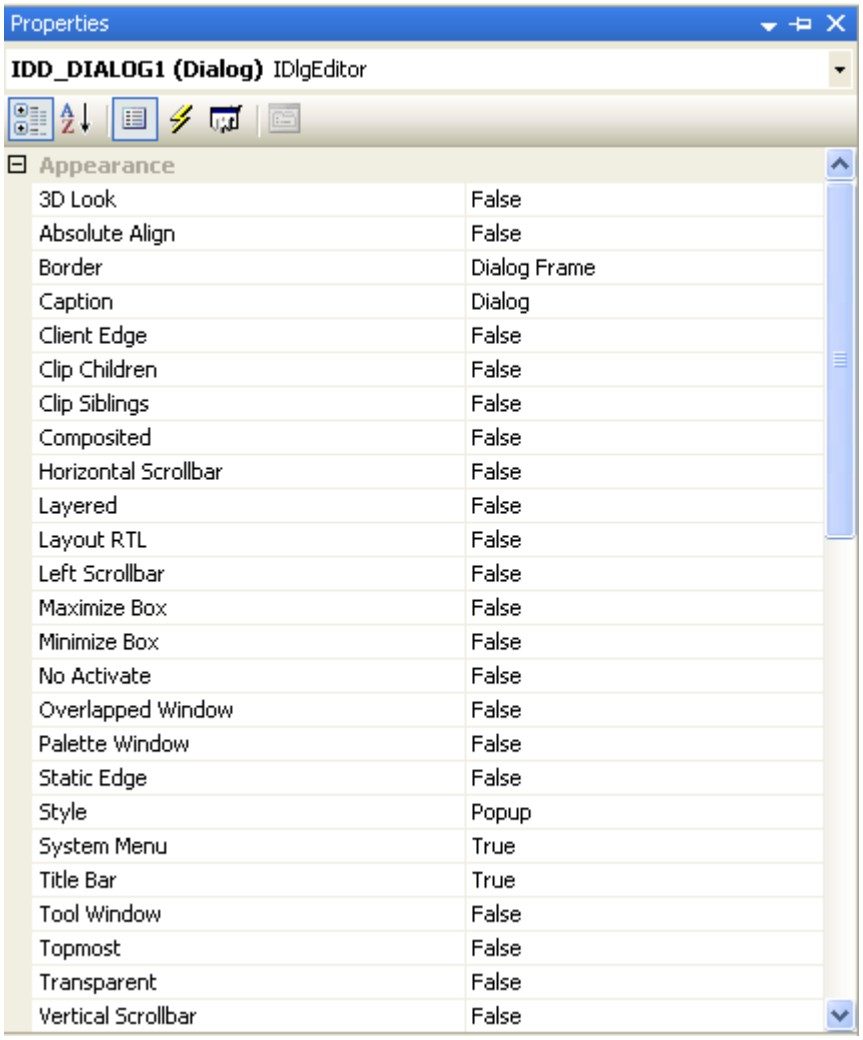
После добавления диалога в ресурсы приложения шаблон диалогового окна будет определён в файле описания ресурсов. При этом диалогу будет назначен идентификатор (например, **IDD_DIALOG1**), который впоследствии можно изменить на идентификатор, отражающий семантику ресурса.

Рекомендуется ознакомить слушателей с некоторыми свойствами диалога. Для активизации окна свойств диалога следует выбрать **View -> Other Windows -> Properties Window (либо <Alt><Enter>).**

В частности, рассмотреть со слушателями следующие свойства:

- предоставление пользователю возможности изменения размеров диалогового окна (свойство **Border** со значением **Resizing**);
- возможность вывода названия программы в заголовок окна (свойство **Caption**);

- наличие кнопки минимизации окна (свойство **Minimize Box** со значением **True**);
- наличие кнопки полноэкранной развёртки окна (свойство **Maximize Box** со значением **True**);
- наличие системного меню (свойство **System Menu** со значением **True**);
- наличие строки заголовка (свойство **Title Bar** со значением **True**);
- возможность расположения окна диалога в центре экрана при запуске программы (свойство **Center** со значением **True**).



3.1. Создание приложения на основе модального диалога

Приложение на основе модального диалога должно содержать как минимум две функции:

- **WinMain** — главную функцию, в которой создается модальное диалоговое окно программы;
- **DlgProc** — диалоговую процедуру, обеспечивающую обработку сообщений для диалогового окна программы.

```
#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)
{
      // создаём главное окно приложения на основе модального диалога
      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); // закрываем модальный диалог
                  return TRUE;
      }
      return FALSE;
}
```

Необходимо отметить, что вышеприведенный пример не показывает механизм взаимодействия с модальным диалоговым окном (об этом речь пойдёт в последующих занятиях), а лишь демонстрирует способ создания приложения на базе модального диалогового окна.

Создание модального диалога осуществляется функцией API **DialogBox**:

```
INT_PTR DialogBox(
    HINSTANCE hInstance, // дескриптор экземпляра приложения, содержащего
      // шаблон диалогового окна
    LPCTSTR lpTemplate, // указатель на строку, содержащую имя шаблона
      // диалогового окна
      HWND hWndParent, // дескриптор родительского окна
    DLGPROC lpDialogFunc // указатель на диалоговую процедуру
);
```

Особенность работы данной функции заключается в том, что она не возвращает управление до тех пор, пока функция обратного вызова (диалоговая процедура) не закроет модальное диалоговое окно, вызвав функцию API **EndDialog**:

```
BOOL EndDialog(
      HWND hDlg, // дескриптор диалогового окна
      INT_PTR nResult // значение, возвращаемое функцией DialogBox
);
```

Следует отметить, что диалоговая процедура во многом напоминает оконную процедуру. Она должна иметь спецификатор **CALLBACK**, поскольку вызывается операционной системой. Имя функции может быть произвольным. Диалоговая процедура принимает такой же набор параметров, что и обычная оконная процедура. Однако существуют некоторые различия между диалоговой процедурой и оконной процедурой.

- Оконная процедура возвращает значение типа **LRESULT**, а диалоговая процедура — значение типа **BOOL**.
- Если оконная процедура не обрабатывает какое-то сообщение, то она вызывает функцию **DefWindowProc**. Если диалоговая процедура не обрабатывает какое-то сообщение, то она возвращает значение **FALSE** (вызов стандартного обработчика сообщения).
- Если оконная процедура обрабатывает какое-то сообщение, то она возвращает значение **0.** Если диалоговая процедура обрабатывает

какое-то сообщение, то она возвращает значение **TRUE** (запрет вызова стандартного обработчика сообщения).

- Для закрытия обычного приложения (не диалогового!!!) в оконной процедуре следует обработать сообщение **WM_DESTROY**. Для закрытия диалогового приложения необходимо в диалоговой процедуре обработать сообщение **WM_CLOSE**.

3.2. Создание приложения на основе немодального диалога

Немодальные диалоговые окна создаются с помощью функции API **CreateDialog**:

```
HWND CreateDialog(
      HINSTANCE hInstance, // дескриптор экземпляра приложения, содержащего
      // шаблон диалогового окна
      LPCTSTR lpTemplate, // указатель на строку, содержащую имя шаблона
      // диалогового окна
      HWND hWndParent, // дескриптор родительского окна
      DLGPROC lpDialogFunc // указатель на диалоговую процедуру
);
```

Функция **CreateDialog** в отличие от функции **DialogBox** сразу же возвращает дескриптор диалогового окна, не дожидаясь его закрытия.

Для того чтобы немодальное окно появилось на экране необходимо в свойствах шаблона диалогового окна установить значение **True** для свойства **Visible**. Другим способом отображения немодального окна является вызов функции API **ShowWindow**.

Кроме того, в теле функции **WinMain** следует предусмотреть цикл обработки сообщений – ещё одно отличие от модального диалогового окна.

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

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

int WINAPI _tWinMain(HINSTANCE hInst, HINSTANCE hPrev, LPTSTR lpszCmdLine,
                     int nCmdShow)
{
      MSG msg;
      // создаём главное окно приложения на основе немодального диалога
      HWND hDialog = CreateDialog(hInst, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                                  DlgProc);
      // Отображаем окно
      ShowWindow(hDialog, nCmdShow);


      //Запускаем цикл обработки сообщений
      while(GetMessage(&msg, 0, 0, 0))
      {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
      }
      return msg.wParam;
}

BOOL CALLBACK DlgProc(HWND hWnd, UINT mes, WPARAM wp, LPARAM lp)
{
      switch(mes)
      {
            case WM_CLOSE:
                  // закрываем немодальный диалог
                  DestroyWindow(hWnd); // разрушаем окно
                  // останавливаем цикл обработки сообщений
                  PostQuitMessage(0);
                  return TRUE;
      }
      return FALSE;
}
```

Необходимо отметить, что вышеприведенный пример не показывает механизм взаимодействия с немодальным диалоговым окном (об этом речь пойдёт в последующих занятиях), а лишь демонстрирует способ создания приложения на базе немодального диалогового окна.

Как видно из вышеприведенного кода, для закрытия приложения, созданного на основе немодального диалога, необходимо в диалоговой процедуре при обработке сообщения **WM_CLOSE** предусмотреть вызов функции **API DestroyWindow** для разрушения диалогового окна, и кроме того, остановить цикл обработки сообщений с помощью функции **API PostQuitMessage.**

```
BOOL DestroyWindow(HWND hWnd);
```

4. Общие сведения об элементах управления

Как было отмечено ранее, диалоговое приложение взаимодействует с пользователем посредством одного или нескольких элементов управления. Элементы управления выполняют основную функциональную нагрузку в диалоговом окне, которое является для них родительским. Windows поддерживает так называемые **базовые элементы управления**, включая кнопки (**Button**), флажки (**Check Box**), переключатели (**Radio Button**), списки (**List Box**), окна ввода (**Edit Control**), комбинированные списки (**Combo Box**), статические элементы (**Static Text**), полосы прокрутки (**Scroll Bar**), рамки (**Group Box**).

Помимо базовых элементов управления, которые поддерживались самыми ранними версиями Windows, в системе используется библиотека элементов управления общего пользования (**common control library**). Общие элементы управления, включенные в эту библиотеку, дополняют базовые элементы управления и позволяют придать приложениям более совершенный вид. К общим элементам управления относятся панель инструментов (**Toolbar**), окно подсказки (**Tooltip**), индикатор (**Progress Control**), счётчик (**Spin Control**), строка состояния (**Status Bar**) и другие.

Обычно элементы управления определяются в шаблоне диалогового окна на языке описания шаблона диалога. Одним из атрибутов описания элемента управления в шаблоне диалога является **идентификатор элемента управления.**

Каждый элемент управления, описанный в шаблоне диалога, реализуется Windows в виде окна соответствующего класса. Например, все кнопки относятся к классу **BUTTON**. Примерами других предопределён-

ных классов для элементов управления являются **STATIC**, **LISTBOX**, **EDIT**, **COMBOBOX** и другие.

Как упоминалось ранее, элемент управления является дочерним окном по отношению к диалоговому окну. Как любое окно, элемент управления идентифицируется своим дескриптором типа **HWND**. Однако если элемент управления определен в шаблоне диалога, то приложению известен только его идентификатор. В то же время многие функции, работающие с элементом управления, принимают в качестве параметра его дескриптор. Для получения дескриптора элемента управления по его идентификатору используется функция API **GetDlgItem**:

```
HWND GetDlgItem(
       HWND hDlg, // дескриптор диалогового окна
       int nIDDlgItem  // идентификатор элемента управления
 );
```

В некоторых случаях возникает необходимость получения идентификатора элемента управления по его дескриптору. Для этого используется функция API **GetDlgCtrlID**:

```
int GetDlgCtrlID(
       HWND hwndCtrl // дескриптор элемента управления
   );
```

Чаще всего элементы управления определяются в шаблоне диалогового окна. Существует, однако, и альтернативный способ создания и размещения элемента управления при помощи функции **CreateWindowEx**, рассмотренной на одном из предыдущих занятий. В этом случае во втором параметре функции передается имя предопределенного оконного класса.

Элементы управления могут быть **разрешенными** (**enabled**) или **запрещенными** (**disabled**). По умолчанию все элементы управления имеют статус разрешенных элементов. Запрещенные элементы выводят-

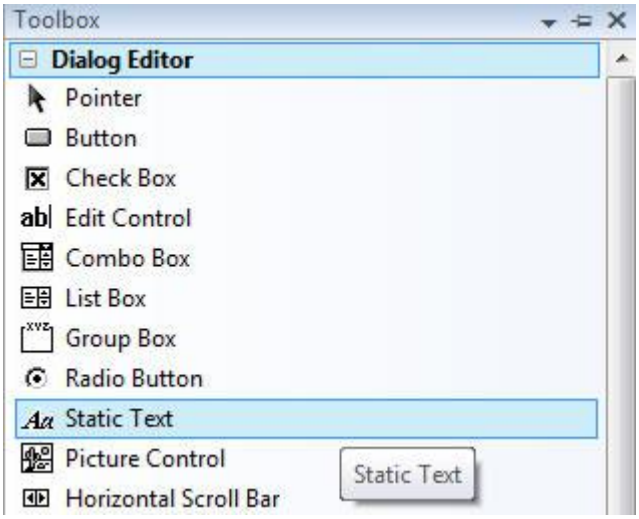
ся на экран серым цветом и не воспринимают пользовательский ввод с клавиатуры или от мыши. Изменение статуса элементов управления осуществляется при помощи функции API **EnableWindow**:

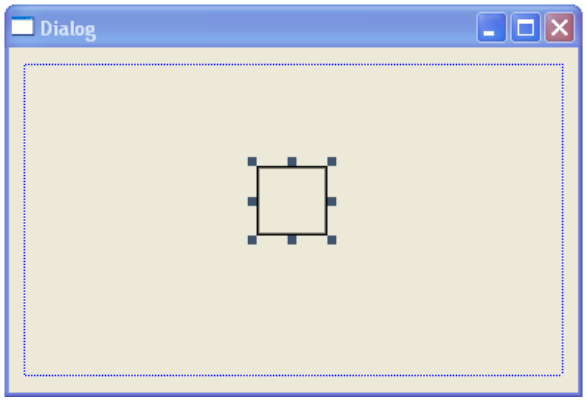
```
BOOL EnableWindow(
      HWND hWnd, // дескриптор окна
      BOOL bEnable // если данный параметр равен TRUE, то окно будет
      // разрешенным, в противном случае - запрещенным
);
```

4.1. Статический элемент управления Static Text

Статический элемент управления **StaticText** представляет собой средство описания чего-либо в диалоге и чаще всего просто отображается в виде текста.

Чтобы разместить на диалоге статический элемент управления следует активизировать окно **Toolbox** (<**Ctrl**><**Alt**><**X**>) и «перетащить» элемент управления на форму диалога.

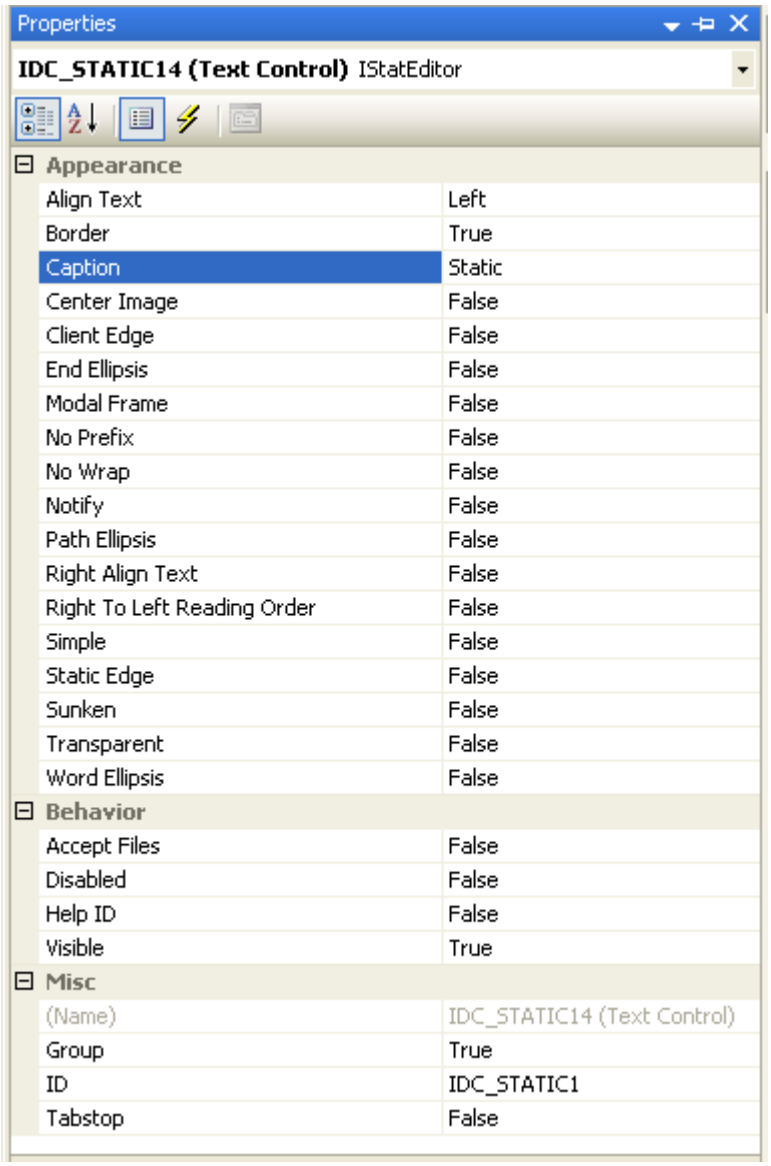




Акцентировать внимание слушателей на том, что обычно всем статическим элементам управления назначается идентификатор **IDC_STATIC**:

```
#define IDC_STATIC (-1)
```

Однако, если к «статику» необходимо будет обращаться в коде приложении, то ему следует назначить уникальный идентификатор, например **IDC_STATIC1**.



Рекомендуется ознакомить слушателей с некоторыми свойствами статического элемента управления **Static Text**.

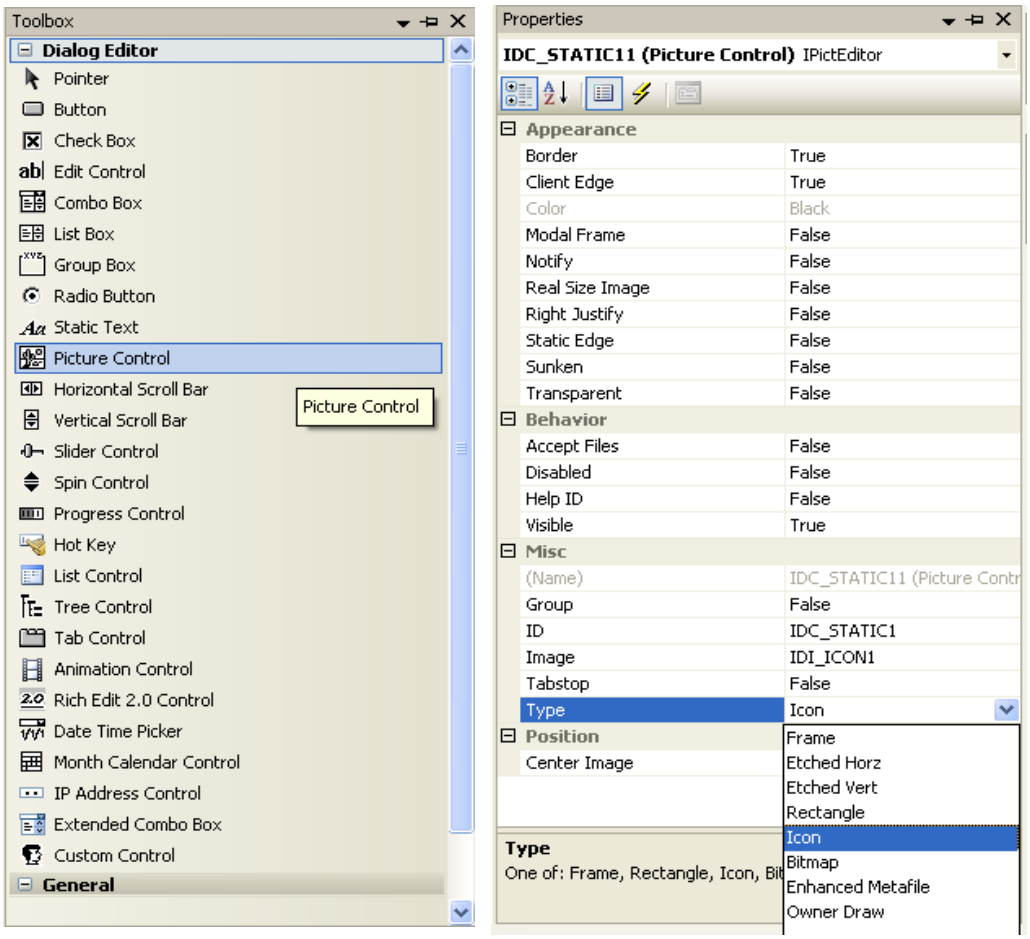
- Свойство **Caption** содержит текстовую строку, которая будет отображаться внутри ограничивающего прямоугольника элемента **Static Text**. При этом в строке могут использоваться управляющие символы **\t** (табуляция) и **\n** (перевод строки).

- Свойство **Border** со значением **True** позволяет установить тонкую рамку вокруг элемента управления.
- Свойство **Align text** позволяет задать выравнивание текста по горизонтали. Можно использовать значение **Left** (по умолчанию), **Center** или **Right**.
- Свойство **Center Image** позволяет задать центрирование текста по вертикали.

4.2. Статический элемент управления Picture Ctrl

Статический элемент управления **PictureCtrl** предназначен для размещения изображения на диалоговом окне.

Прежде чем поместить изображение (иконку или растровый образ) на форму диалога, необходимо сначала включить его в состав ресурсов проекта. После этого добавить на форму диалога элемент управления **PictureCtrl**.



В свойстве **Type** элемента управления следует задать тип изображения, например, **Icon**. В этом случае свойство **Image** станет доступным, что позволит выбрать из выпадающего списка нужный идентификатор изображения (например, **IDI_ICON1**).

В качестве примера, демонстрирующего использование статических элементов управления, привести следующий код:

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

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


HWND hStatic1, hStatic2;
TCHAR szCoordinates[20];
HINSTANCE hInst;
const int LEFT = 15, TOP = 110, WIDTH = 380, HEIGHT = 50;

int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,
                  LPTSTR lpszCmdLine, int nCmdShow)
{
      hInst = hInstance;
      // создаём главное окно приложения на основе модального диалога
      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); // закрываем модальный диалог
                  return TRUE;
            // WM_INITDIALOG - данное сообщение приходит после создания
            // диалогового окна, но перед его отображением на экран
            case WM_INITDIALOG:
                  // получаем дескриптор статика, размещенного на диалоге
                  hStatic1 = GetDlgItem(hWnd, IDC_STATIC1);
                  //создаём статик с помощью CreateWindowEx
                  hStatic2 = CreateWindowEx(0, TEXT("STATIC"), 0,
                              WS_CHILD | WS_VISIBLE | WS_BORDER | SS_CENTER |
                              WS_EX_CLIENTEDGE, LEFT, TOP, WIDTH, HEIGHT,
                              hWnd, 0, hInst, 0);
                  return TRUE;
            case WM_MOUSEMOVE:
                  // текущие координаты курсора мыши
                  wsprintf(szCoordinates, TEXT("X=%d  Y=%d"),
                              LOWORD(lParam), HIWORD(lParam));
                  // строка выводится на статик
                  SetWindowText(hStatic1, szCoordinates);
                  return TRUE;
```

```
            case WM_LBUTTONDOWN:
                  SetWindowText(hStatic2, TEXT("Нажата левая кнопка мыши"));
                  return TRUE;
            case WM_RBUTTONDOWN:
                  SetWindowText(hStatic2, TEXT("Нажата правая кнопка мыши"));
                  return TRUE;
      }
      return FALSE;
}
```

Акцентировать внимание слушателей на сообщении **WM_INITDIALOG**, которое первым приходит в диалоговую процедуру. Отметить, что сообщение **WM_INITDIALOG** приходит после создания диалогового окна, но перед показом на экран. В обработчике этого сообщения удобно инициализировать элементы управления, а также выполнять другие инициализирующие действия.

В данном приложении был использован ещё один статический элемент управления – **Group Box**, который представляет собой рамку с заголовком. Этот элемент обычно используется для визуального группирования других элементов и может содержать заголовок (название) группы.

5. Практическая часть

Создать диалоговое приложение, позволяющее перемещать и пропорционально изменять размеры статика, расположенного на форме диалога. Для перемещения статика следует использовать клавиши управления курсором, а для пропорционального изменения размеров статика – клавиши **«+»** и **«-»**.

6. Подведение итогов

Подвести общие итоги занятия. Ещё раз отметить основное предназначение диалогового окна. Напомнить слушателям, чем отличается мо-

дальный диалог от немодального диалога. Акцентировать внимание слушателей на наиболее тонких моментах разработки приложения на основе модального и немодального диалогов. Отметить основные отличия между оконной процедурой и диалоговой процедурой при обработке сообщений.

7. Домашнее задание

Разработать приложение, созданное на основе диалогового окна, и обладающее следующей функциональностью.

- Пользователь «щелкает» левой кнопкой мыши по форме диалога и, не отпуская кнопку, ведёт по ней мышку, а в момент отпускания кнопки по полученным координатам прямоугольника (как известно, двух точек на плоскости достаточно для создания прямоугольника) создаётся «статик», который содержит свой порядковый номер (имеется в виду порядок появления «статика» на форме).
- Минимальный размер «статика» составляет 10x10, а при попытке создания элемента управления меньших размеров пользователь должен увидеть соответствующее предупреждение.
- При щелчке правой кнопкой мыши над поверхностью «статика» в заголовке окна должна появиться информация о статике (порядковый номер «статика», ширина и высота, а также координаты «статика» относительно родительского окна). В случае если в точке щелчка находится несколько «статиков», то предпочтение отдается «статику» с наибольшим порядковым номером.

- При двойном щелчке левой кнопки мыши над поверхностью «статика» он должен исчезнуть с формы (для этого можно воспользоваться функцией **DestroyWindow**, вызывая её для соответствующего объекта «статика»). В случае если в точке щелчка находится несколько «статиков», то предпочтение отдается «статику» с наименьшим порядковым номером.

При разработке приложения рекомендуется использовать библиотеку STL.