Visual C++

ШАГ компьютерная академия

Разработка Windows приложений с использованием VC++ и WinAPI

Урок №7

Меню

Содержание

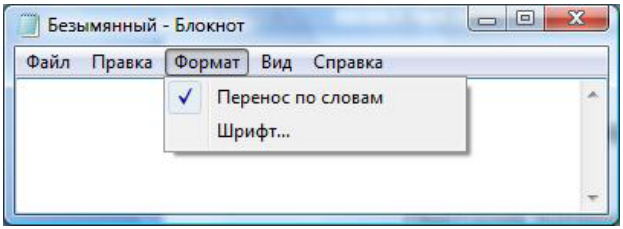
1. Меню. Общее понятие. Виды меню

Меню представляет собой список команд, позволяющих выполнить любую операцию, реализованную приложением. Различают следующие виды меню:

- главное меню;
- системное меню;
- контекстное меню.

1.1. Главное меню

Меню, располагающееся ниже заголовка окна приложения, называется ***главным меню*** (**main menu**), или ***меню верхнего уровня***. Главное меню относится ко всему приложению.

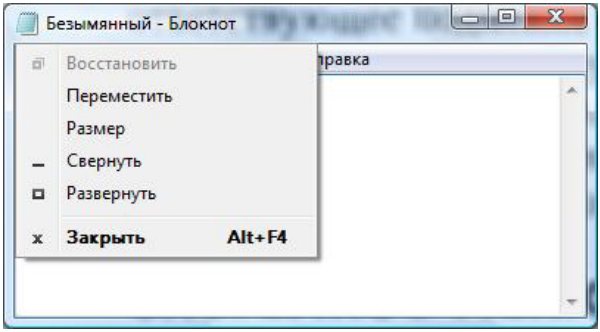


При выборе одного из пунктов главного меню, как правило, активизируется ***раздел меню***. Раздел меню представляет собой ***выпадающее меню*** («**popup menu**») или ***подменю***. Например, на представленном выше изображении пункт главного меню «**Формат**» имеет подменю, состоящее из двух пунктов: «**Перенос по словам**» и «**Шрифт...**» Троеточие

в конце названия пункта меню означает, что при выборе пункта меню появится дополнительное диалоговое окно для задания определённых настроек.

1.2. Системное меню

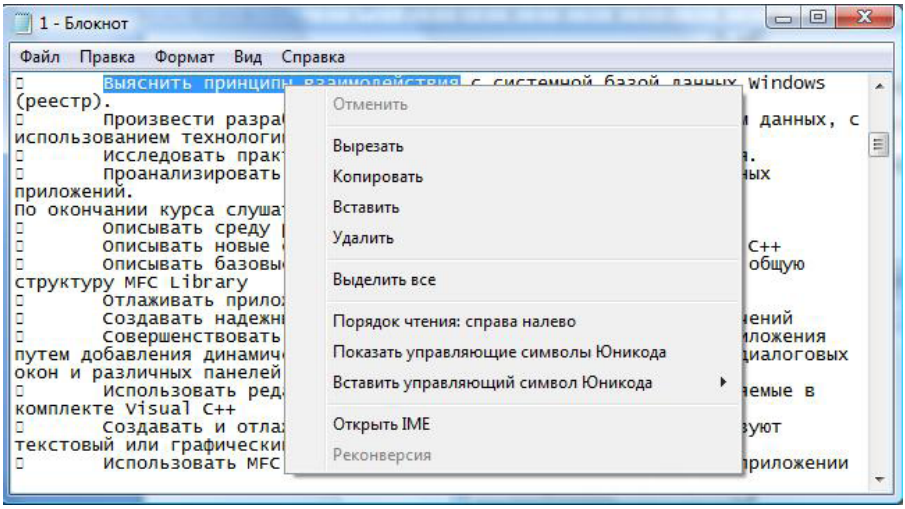
Каждое окно, имеющее заголовок, может предоставлять доступ к системному меню, которое вызывается щелчком левой кнопки мыши на иконке, расположенной в левой части заголовка окна. Системное меню открывается как подменю и обладает набором стандартных команд.



1.3. Контекстное меню

Контекстное меню (меню быстрого вызова или «*shortcut menu*») — это меню, которое появляется в любой части окна приложения при щелчке правой кнопкой мыши. Оно выглядит как выпадающее меню, но без привязки к меню верхнего уровня.

Обычно содержание контекстного меню изменяется в зависимости от «*контекста*» — места, где произведен щелчок правой кнопкой мыши. В сложных приложениях контекстные меню часто содержат пункты, дублирующие

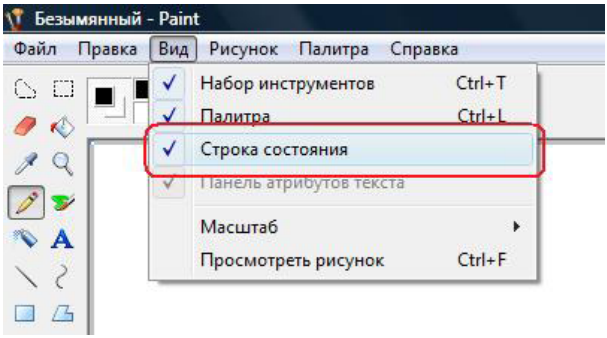


команды основного меню, но сгруппированные иначе, чтобы максимально облегчить пользователю работу с приложением. После выбора пункта контекстного меню оно исчезает с экрана.

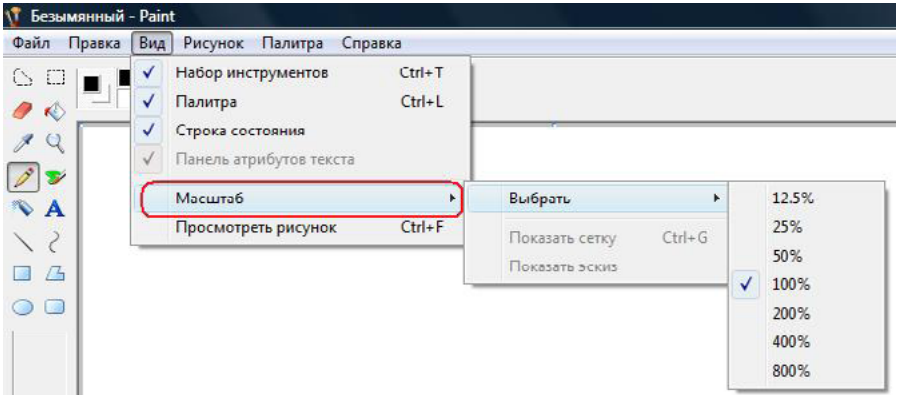
2. Типы и состояния пунктов меню

Различают два типа пунктов меню:

- пункт меню «команда»;
- пункт меню «подменю».



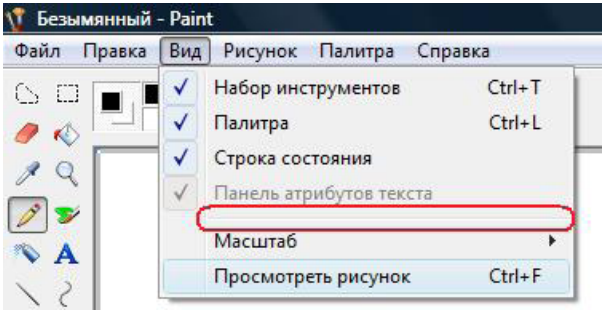
Пункт меню «**команда**» — это конечный пункт на иерархическом дереве меню. При выборе такого пункта меню обычно выполняется некоторое действие. При этом



в программном коде пункту меню «команда» назначается уникальный целочисленный идентификатор.

Пункт меню **«подменю»** — это заголовок выпадающего меню более низкого уровня.

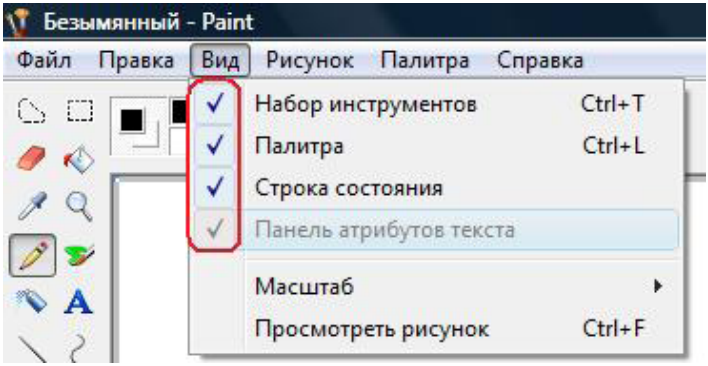
Следует отметить, что восприятие подменю сбольшим количеством пунктов облегчается, если они разделены на некоторые логические группы спомощью сепараторов. ***Сепаратор*** — это элемент меню, который представляет собою горизонтальную разделительную линию.



Пункты меню могут быть ***разрешенными*** (**enabled**), ***запрещенными*** (**disabled**) и ***недоступными*** (**grayed**). По умолчанию пункт меню является разрешенным. Когда выбирается такой пункт, система посылает оконной процедуре сообщение **WM_COMMAND** или отображает соответствующее подменю, в зависимости от типа пункта.

Запрещенный и недоступный пункты сточки зрения поведения одинаковы. Их можно выделить, но нельзя выбрать, т.е. при щелчке мышью или при нажатии клавиши **<Enter>** ничего не происходит. Различаются запрещенный и недоступный пункты только внешним видом. Запрещенный пункт выглядит точно так же, как разрешенный, а недоступный пункт меню отображается серым цветом.

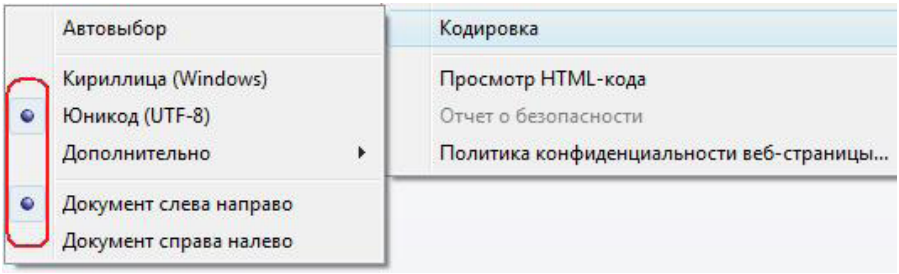
Необходимо отметить, что в хорошо продуманном интерфейсе пользователя приложение должно управлять статусом пунктов меню в зависимости от текущего состояния программы. Например, команда копирования текста в буфер обмена Windows не имеет смысла, если не выделен фрагмент текста в поле ввода. В такой ситуации лучше отменить и выделить серым цветом соответствующий пункт меню. Вообще, рекомендуется делать недоступными те пункты меню, использование которых в данный момент бессмысленно или даже небезопасно с точки зрения устойчивости работы приложения.



Иногда пункт меню может использоваться в роли флажка (**check box**). Флажок может быть установлен или сброшен. При этом переход из одного состояния в другое происходит при каждом выборе пункта меню.

Флажки обычно объединяются в группы и обеспечивают выбор либо одной, либо нескольких опций одновременно.

Пункты меню могут использоваться также в роли переключателей (**radio button**). Переключатели, как и флажки,



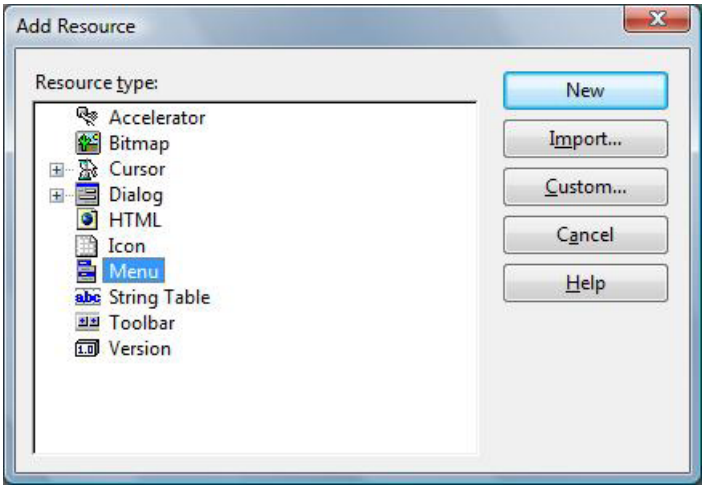
обычно используются в группе. В отличие от флажков, переключатели связываются только с взаимоисключающими опциями, поэтому в группе можно выбрать только один переключатель.

3. Создание меню с помощью средств интегрированной среды разработки приложений Microsoft Visual Studio

Включение меню в приложение требует следующих шагов:

- определение шаблона меню в файле описания ресурсов;
- загрузка меню при создании главного окна;
- обработка событий меню.

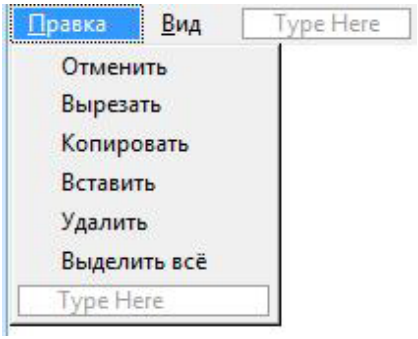
Для того чтобы определить меню в файле описания ресурсов необходимо активизировать вкладку **Resource View**, в которой с помощью контекстного меню вызвать диалог добавления ресурса **Add -> Resource...**



В этом диалоговом окне необходимо выбрать из списка **Menu** и нажать кнопку **New**, в результате чего будет открыто окно редактора меню с заготовкой полосы нового меню.

По умолчанию редактор присваивает первому из создаваемых шаблонов меню идентификатор **IDR_MENU1**.

Впоследствии этот идентификатор можно изменить на другой идентификатор, отражающий семантику ресурса.

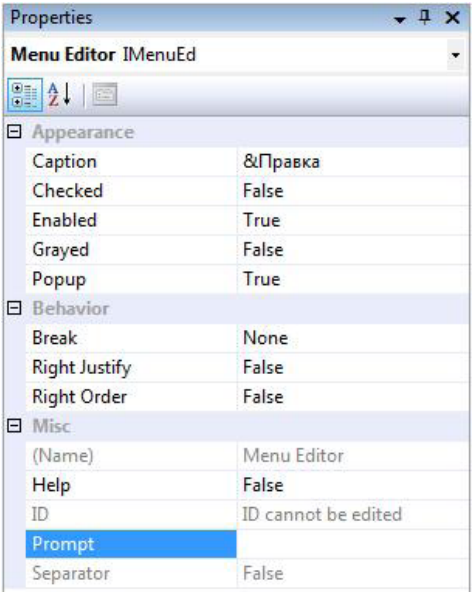


Рассмотрим некоторые свойства меню.

- Свойство **Caption** определяет название пункта меню. Если в названии встречается символ «**&**», то следующий за ним символ является мнемоническим и будет выделен подчёркиванием при нажатии клавиши **<Alt>**. Это позволяет определить горячую клавишу (**hotkey**) для быстрого способа выбора пункта при помощи

клавиатуры. Например, на представленном выше изображении пункт меню «**Правка**» имеет мнемонический символ **<П>**, что позволяет выбрать пункт, используя комбинацию клавиш **<Alt><П>**.

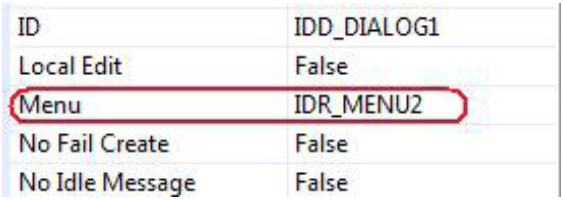
- Свойство **ID** определяет идентификатор пункта меню (разумеется, только в том случае, если пункт меню является командой, а не «**popup**»).
- Свойство **Checked** определяет, будет ли отмечен пункт меню.



- Свойство **Grayed** обуславливает недоступность пункта меню в исходном состоянии.
- Свойство **Popup** при значении True определяет подменю. В противном случае пункт меню является обычной командой.

- Свойство **Break** может принимать одно из трех значений:
  - **None** — обычный пункт меню;
  - **Column** — для меню верхнего уровня пункт выводится с новой строки, а для подменю — в новом столбце;
  - **Bar** — дополнительный столбец подменю отделяется вертикальной линией.
- Свойство **Separator** задаёт горизонтальную разделительную линию.

После определения меню в файле описания ресурсов оно ещё не появится в составе главного окна приложения. Для этого необходимо присоединить меню к окну. Один из способов присоединения меню к окну — указание идентификатора меню в свойстве Menu диалогового окна.



Существует альтернативный программный способ присоединения меню к главному окну приложения. Для этого сначала необходимо загрузить меню из ресурсов приложения, используя функцию API **LoadMenu**:

```
HMENU LoadMenu(
    HINSTANCE hInstance, // дескриптор приложения
    LPCTSTR lpMenuName // указатель на строку, содержащую
имя меню в ресурсах
);
```

После загрузки меню, его необходимо установить с помощью функции API **SetMenu**:

```
BOOL SetMenu(
    HWND hWnd, //дескриптор окна, к которому
               //присоединяется меню
    HMENU hMenu //дескриптор меню
);
```

Следующий фрагмент кода демонстрирует программный способ присоединения меню к диалоговому окну приложения.

```
//Загрузим меню из ресурсов приложения
HMENU hMenu = LoadMenu(GetModuleHandle(NULL),
              MAKEINTRESOURCE(IDR_MENU2));
//Присоединим меню к главному окну приложения
SetMenu(hDialog, hMenu);
```

Следует также отметить, что Win API предоставляет функции, позволяющие получить дескриптор, как для главного меню, так и для любого подменю.

Для получения дескриптора главного меню служит функция API **GetMenu**.

```
HMENU GetMenu(
    HWND hWnd //дескриптор главного окна приложения
);
```

Для получения дескриптора подменю служит функция API **GetSubMenu**.

```
HMENU GetSubMenu (
HMENU hMenu, //дескриптор родительского меню
int nPos //позиция пункта-подменю в родительском меню
);
```

4. Обработка сообщений меню

Как было отмечено ранее, операционная система Windows посылает сообщение **WM_COMMAND** при каждом выборе пункта меню, определяющего команду. При этом **LOWORD(wParam)** содержит идентификатор пункта меню, а **HIWORD(wParam)** и **lParam** содержат нулевые значения.

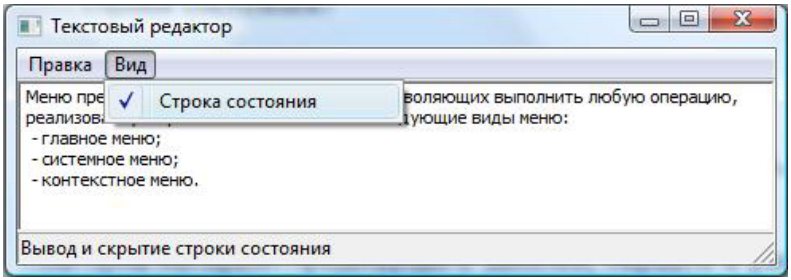
Чаще всего **WM_COMMAND** — единственное сообщение, обрабатываемое приложением, которое может поступить от меню. При выборе пунктов системного меню вместо указанного сообщения отправляется сообщение **WM_SYSCOMMAND**.

Иногда в программе может потребоваться обработка сообщений **WM_INITMENU** и **WM_INITMENUPOPUP**. Они отправляются непосредственно перед активизацией главного меню или выпадающего меню. Эти сообщения позволяют приложению изменить меню перед тем, как оно будет отображено на экране.

При навигации по меню система отправляет также сообщение **WM_MENUSELECT**. Оно более универсально по сравнению с **WM_COMMAND**, так как инициируется даже тогда, когда выделен недоступный или запрещенный пункт меню. Это сообщение может использоваться для формирования контекстной справки меню, которая отображается в строке состояния приложения.

Для демонстрации техники использования меню рассмотрим приложение, в котором обрабатываются описанные

выше сообщения меню (исходный код приложения прикреплен к PDF-файлу урока, папка **SOURCE/ Menu**).




```
                        //header.h
#pragma once
#include <windows.h>
#include <windowsX.h>
#include <tchar.h>
#include <commctrl.h>
#include "resource.h"
#pragma comment(lib,"comctl32")

                      // MenuBarDlg.h
#pragma once
#include "header.h"

class CMenuBarDlg
{
public:
    CMenuBarDlg(void);
public:
    static BOOL CALLBACK DlgProc(HWND hWnd, UINT mes,
        WPARAM wp, LPARAM lp);
    static CMenuBarDlg* 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_OnSize(HWND hwnd, UINT state, int cx, int cy);
    void Cls_OnInitMenuPopup(HWND hwnd, HMENU hMenu,
        UINT item, BOOL fSystemMenu);
    void Cls_OnMenuSelect(HWND hwnd, HMENU hmenu, int item,
        HMENU hmenuPopup, UINT flags);
    HWND hDialog, hStatus, hEdit;
    BOOL bShowStatusBar;
};

                        //MenuBarDlg.cpp
#include "MenuBarDlg.h"

CMenuBarDlg* CMenuBarDlg::ptr = NULL;

CMenuBarDlg::CMenuBarDlg(void)
{
    ptr = this;
    bShowStatusBar = TRUE;
}

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

BOOL CMenuBarDlg::Cls_OnInitDialog(HWND hwnd,
        HWND hwndFocus, LPARAM lParam)
{
    hDialog = hwnd;
    //Получим дескриптор текстового поля
    hEdit = GetDlgItem(hDialog, IDC_EDIT1);
    //Создадим строку состояния
    hStatus = CreateStatusWindow(WS_CHILD | WS_VISIBLE |
        CCS_BOTTOM | SBARS_TOOLTIPS | SBARS_SIZEGRIP, 0,
        hDialog, WM_USER);
```

```
    //Загрузим меню из ресурсов приложения
    HMENU hMenu = LoadMenu(GetModuleHandle(NULL),
        MAKEINTRESOURCE(IDR_MENU2));
    //Присоединим меню к главному окну приложения
    SetMenu(hDialog, hMenu);
    return TRUE;
}

//Обработчик сообщения WM_COMMAND будет вызван
//при выборе пункта меню
void CMenuBarDlg::Cls_OnCommand(HWND hwnd, int id,
        HWND hwndCtl, UINT codeNotify)
{

    switch(id)
    {
    case ID_UNDO:
        //Отменим последнее действие
        SendMessage(hEdit, WM_UNDO, 0, 0);
        break;
    case ID_CUT:
        //Удалим выделенный фрагмент текста в буфер обмена
        SendMessage(hEdit, WM_CUT, 0, 0);
        break;
    case ID_COPY:
        //Скопируем выделенный фрагмент текста в буфер обмена
        SendMessage(hEdit, WM_COPY, 0, 0);
        break;
    case ID_PASTE:
        //Вставим текст в Edit Control из буфера обмена
        SendMessage(hEdit, WM_PASTE, 0, 0);
        break;
    case ID_DEL:
        //Удалим выделенный фрагмент текста
        SendMessage(hEdit, WM_CLEAR, 0, 0);
        break;
    case ID_SELECTALL:
```

```
        //Выделим весь текст в Edit Control
        SendMessage(hEdit, EM_SETSEL, 0, -1);
        break;
    case ID_STATUSBAR:
        //Если флаг равен TRUE, то строка состояния
        //отображена
        if(bShowStatusBar)
        {
            //Получим дескриптор главного меню
            HMENU hMenu = GetMenu(hDialog);
            //Снимем отметку с пункта меню
            //"Строка состояния"
            CheckMenuItem(hMenu, ID_STATUSBAR,
                MF_BYCOMMAND | MF_UNCHECKED);
            //Скроем строку состояния
            ShowWindow(hStatus, SW_HIDE);
        }
        else
        {
            //Получим дескриптор главного меню
            HMENU hMenu = GetMenu(hDialog);
            //Установим отметку на пункте меню
            //"Строка состояния"
            CheckMenuItem(hMenu,ID_STATUSBAR, MF_BYCOMMAND |
                MF_CHECKED);
            //Отобразим строку состояния
            ShowWindow(hStatus, SW_SHOW);
        }
        bShowStatusBar = !bShowStatusBar;
    }
}

//Обработчик сообщения WM_SIZE будет вызван при изменении
//размеров главного окна  либо при сворачивании
//восстановлении главного окна
void CMenuBarDlg::Cls_OnSize(HWND hwnd, UINT state,
int cx, int cy)
```

```
{
    RECT rect1, rect2;
    //Получим координаты клиентской области главного окна
    GetClientRect(hDialog, &rect1);
    //Получим координаты единственной секции строки
     //состояния
    SendMessage(hStatus, SB_GETRECT, 0, (LPARAM)&rect2);
    //Установим новые размеры текстового поля
    MoveWindow(hEdit, rect1.left, rect1.top, rect1.right,
         rect1.bottom - (rect2.bottom - rect2.top), 1);
    //Установим размер строки состояния,
    //равный ширине клиентской области главного окна
    SendMessage(hStatus, WM_SIZE, 0, 0);
}

//Обработчик WM_INITMENUPOPUP будет вызван непосредственно
//перед активизацией всплывающего меню
void CMenuBarDlg::Cls_OnInitMenuPopup(HWND hwnd,
        HMENU hMenu, UINT item, BOOL fSystemMenu)
{
    if(item == 0) // Активизируется пункт меню "Правка"
    {
        //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?
        {
            //Если имеется выделенный текст, то сделаем
            //разрешёнными пункты меню "Копировать",
            //"Вырезать" и "Удалить"
            EnableMenuItem(hMenu, ID_COPY, MF_BYCOMMAND |
                MF_ENABLED);
            EnableMenuItem(hMenu, ID_CUT, MF_BYCOMMAND |
                MF_ENABLED);
            EnableMenuItem(hMenu, ID_DEL, MF_BYCOMMAND |
                MF_ENABLED);
        }
```

```
        else
        {
            //Если отсутствует выделенный текст,
            //то сделаем недоступными пункты меню
            55"Копировать", "Вырезать" и "Удалить"
            EnableMenuItem(hMenu, ID_COPY, MF_BYCOMMAND |
                MF_GRAYED );
            EnableMenuItem(hMenu, ID_CUT, MF_BYCOMMAND |
                MF_GRAYED);
            EnableMenuItem(hMenu, ID_DEL, MF_BYCOMMAND |
                MF_GRAYED);
        }
        //Имеется ли текст в буфере обмена?
        if(IsClipboardFormatAvailable(CF_TEXT))
            //Если имеется текст в буфере обмена,
            //то сделаем разрешённым пункт меню "Вставить"
            EnableMenuItem(hMenu, ID_PASTE, MF_BYCOMMAND |
                MF_ENABLED);
         else
            //Если отсутствует текст в буфере обмена,
            //то сделаем недоступным пункт меню "Вставить"
            EnableMenuItem(hMenu, ID_PASTE, MF_BYCOMMAND |
                MF_GRAYED);
            //Существует ли возможность отмены последнего
            //действия?

            if(SendMessage(hEdit, EM_CANUNDO, 0, 0))
                //Если существует возможность отмены
                //последнего действия,
                //то сделаем разрешённым пункт меню "Отменить"
                EnableMenuItem(hMenu, ID_UNDO, MF_BYCOMMAND
                    | MF_ENABLED);
            else
                //Если отсутствует возможность отмены
                //последнего действия, то сделаем
                //недоступным пункт меню "Отменить"
```

```
                EnableMenuItem(hMenu, ID_UNDO, MF_BYCOMMAND |
                    MF_GRAYED);
                //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)
                //Если не весь текст выделен в Edit Control,
                //то сделаем разрешённым пункт меню
                //"Выделить всё"
                EnableMenuItem(hMenu, ID_SELECTALL,
                    MF_BYCOMMAND | MF_ENABLED);
            else
                //Если выделен весь текст в Edit Control,
                //то сделаем недоступным пункт меню
                //"Выделить всё"
                EnableMenuItem(hMenu, ID_SELECTALL,
                    MF_BYCOMMAND | MF_GRAYED);
    }
}

void CMenuBarDlg::Cls_OnMenuSelect(HWND hwnd, HMENU hmenu,
int item, HMENU hmenuPopup, UINT flags)
{
    //Проверим, является ли выделенный пункт меню заголовком
    //выпадающего подменю?
    if(flags & MF_POPUP)
    {
        //Выделенный пункт меню является заголовком выпадающего
        //подменю
        //Убираем текст со строки состояния
        SendMessage(hStatus, SB_SETTEXT, 0, 0);
    }
    else
    {
        //Выделенный пункт меню является конечным пунктом
        //(пункт меню "команда")
```

```
    TCHAR buf[200];
    //Получим дескриптор текущего экземпляра приложения
    HINSTANCE hInstance = GetModuleHandle(NULL);
    //Зарузим строку из таблицы строк, расположенной
    //в ресурсах приложения
    //При этом идентификатор загружаемой строки строго
    //соответствует идентификатору выделенного пункта меню
    LoadString(hInstance, item, buf, 200);
    //Выводим в строку состояния контекстную справку,
    //соответствующую выделенному пункту меню
    SendMessage(hStatus, SB_SETTEXT, 0, LPARAM(buf));
    }
}

BOOL CALLBACK CMenuBarDlg::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_SIZE, ptr->Cls_OnSize);
        HANDLE_MSG(hwnd, WM_INITMENUPOPUP,
            ptr->Cls_OnInitMenuPopup);
        HANDLE_MSG(hwnd, WM_MENUSELECT,
            ptr->Cls_OnMenuSelect);
    }
    return FALSE;
}

                        //MenuBar.cpp
#include "MenuBarDlg.h"

int WINAPI _tWinMain(HINSTANCE hInst, HINSTANCE hPrev,
LPTSTR lpszCmdLine, int nCmdShow)
{
```

```
    INITCOMMONCONTROLSEX icc = {sizeof(INITCOMMONCONTROLSEX)};
    icc.dwICC = ICC_WIN95_CLASSES;
    InitCommonControlsEx(&icc);
    CMenuBarDlg dlg;
    return DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1),
        NULL, CMenuBarDlg::DlgProc);
}
```

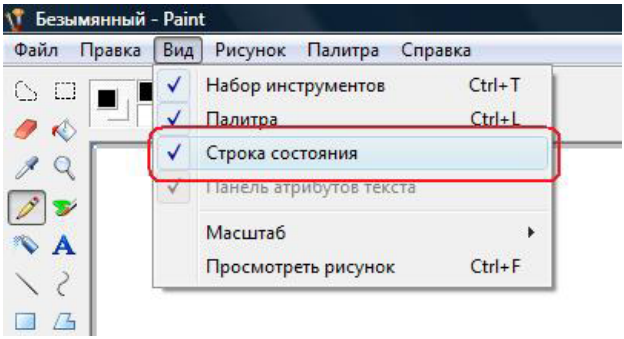
Как видно из вышеприведенного кода приложение обрабатывает сообщение **WM_INITMENUPOPUP**, которое приходит в оконную процедуру диалога непосредственно перед активизацией выпадающего меню. В обработчике этого сообщения наиболее удобно управлять статусом пунктов меню в зависимости от текущего состояния программы. Например, команда удаления текста не имеет смысла, если не выделен фрагмент текста в поле ввода. В этой ситуации было бы логично сделать недоступным соответствующий пункт меню. Таким образом, в обработчике сообщения **WM_INITMENUPOPUP** анализируется текущее состояние текстового поля, а также буфера обмена Windows, вследствие чего те или иные пункты меню «Правка» устанавливаются разрешёнными или недоступными.

Для изменения статуса пунктов меню применяется функция API **EnableMenuItem**.

```
BOOL EnableMenuItem(
    HMENU hMenu, //дескриптор меню
    UINT uIDEnableItem, //идентификатор или позиция пункта
                        //меню
    UINT uEnable //интерпретация второго параметра
                 //и выполняемое действие
);
```

Пункт меню, для которого применяется функция, задается вторым параметром uIDEnableItem. Этому параметру передается либо идентификатор пункта меню, либо позиция пункта меню. Выбор варианта интерпретации задается в третьем параметре. Третий параметр uEnable задается как побитовая операция объединения двух флагов. Первый флаг может содержать одно из следующих значений:

- **MF_BYCOMMAND** — в этом случае второй параметр должен содержать идентификатор пункта меню;
- **MF_BYPOSITION** — в этом случае второй параметр должен содержать относительную позицию пункта меню с отсчетом от нуля. Например, позиция пункта меню «Строка состояния» подменю «Вид» равна 2 (отсчет с нуля!).



Второй флаг может принимать одно из следующих значений:

- **MF_ENABLED** — пункт меню разрешён;
- **MF_DISABLED** — пункт меню запрещён;
- **MF_GRAYED** — пункт меню недоступен.

Следует отметить, если в результате применения функции изменяется статус пункта главного меню, то следует обязательно вызвать функцию API **DrawMenuBar** для повторного отображения изменившейся полосы меню.

```
BOOL DrawMenuBar(
          HWND hWnd //дескриптор главного окна приложения
);
```

В коде приложения также используется функция API **CheckMenuItem**, позволяющая установить или снять отметку на пункте меню.

```
DWORD CheckMenuItem(
    HMENU hMenu, //дескриптор меню
    UINT uIDCheckItem, //идентификатор или позиция пункта меню
    UINT uCheck //интерпретация второго параметра
                //и выполняемое действие
);
```

Второму параметру функции **uIDCheckItem** передается либо идентификатор пункта меню, либо позиция пункта меню. Выбор варианта интерпретации задается в третьем параметре. Третий параметр **uCheck** задается как побитовая операция объединения двух флагов. Первый флаг может содержать одно из следующих значений:

- **MF_BYCOMMAND** — в этом случае второй параметр должен содержать идентификатор пункта меню;
- **MF_BYPOSITION** — в этом случае второй параметр uIDCheckItem должен содержать относительную позицию пункта меню с отсчетом от нуля.

Второй флаг может принимать одно из следующих значений:

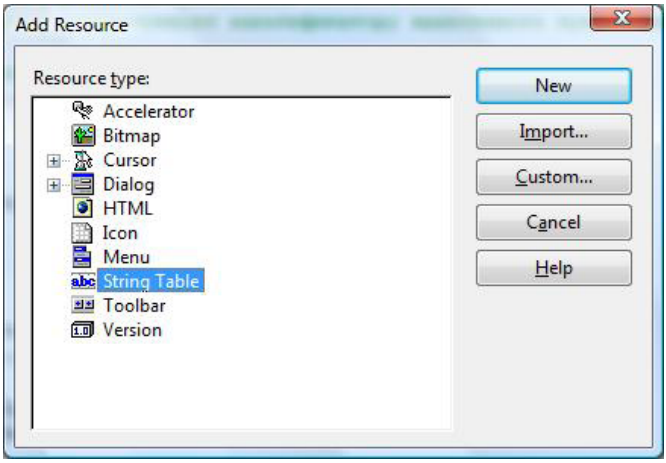
- **MF_CHECKED** — поместить отметку слева от имени пункта меню;
- **MF_UNCHECKED** — снять отметку слева от имени пункта меню.

В функциях **CheckMenuItem** и **EnableMenuItem** рекомендуется использовать флаг **MF_BYCOMMAND**, передавая второму параметру идентификатор пункта меню. Это предотвратит возможные проблемы в процессе сопровождения программы, если потребуется модификация меню, изменяющая относительные позиции пунктов меню.

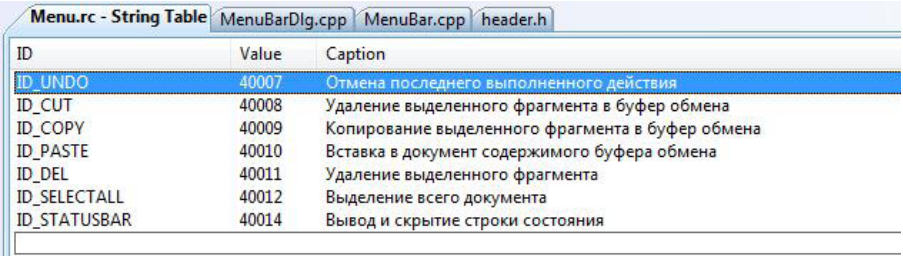
Необходимо отметить, что функцию **CheckMenuItem** можно применять только для пунктов подменю. Пункты главного меню не могут быть помечены.

Продолжая анализировать вышеприведенный код, отметим некоторые особенности при обработке сообщения **WM_MENUSELECT**. Как отмечалось ранее, это сообщение удобно использовать для формирования контекстной справки меню, которая отображается в строке состояния приложения. В данном приложении для отображения текста контекстной справки в строке состояния был использован ресурс таблицы строк. Таблица строк представляет собой список строк, связанных с уникальными целочисленными идентификаторами.

Для того чтобы определить таблицу строк в файле описания ресурсов необходимо активизировать вкладку **Resource View**, в которой с помощью контекстного меню вызвать диалог добавления ресурса **Add -> Resource...**



В этом диалоговом окне следует выбрать из списка **String Table** и нажать кнопку **New**, в результате чего будет открыто окно редактора таблицы строк, в котором необходимо определить список строк, указав для каждой из них целочисленный идентификатор.



ID	Value	Caption
ID_UNDO	40007	Отмена последнего выполненного действия
ID_CUT	40008	Удаление выделенного фрагмента в буфер обмена
ID_COPY	40009	Копирование выделенного фрагмента в буфер обмена
ID_PASTE	40010	Вставка в документ содержимого буфера обмена
ID_DEL	40011	Удаление выделенного фрагмента
ID_SELECTALL	40012	Выделение всего документа
ID_STATUSBAR	40014	Вывод и скрытие строки состояния

Для удобства формирования контекстной справки меню каждой строке сопоставлен уже существующий идентификатор одного из пунктов меню. Это позволяет при обработке сообщения **WM_MENUSELECT** загрузить из ресурсов ту строку, которая соответствует выделенному пункту меню. После этого загруженная строка выводится

в строку состояния, тем самым, обеспечивая контекстную справку для выделенного пункта меню.

Для загрузки строки из ресурса таблицы строк предназначена функция API **LoadString**.

```
int LoadString(
    HINSTANCE hInstance, //дескриптор приложения,
                         //содержащего таблицу строк
    UINT uID,            //идентификатор строки, которая
                         //должна быть загружена
    LPTSTR lpBuffer,     //указатель на буфер, в который
                         //будет записана строка
    int nBufferMax       //размер буфера
);
```

5. Динамическое создание меню. Модификация меню

Рассмотренный ранее способ создания меню с помощью средств интегрированной среды разработки приложений не является единственным. Альтернативным подходом является программное (динамическое) создание меню.

Рассмотрим набор функций API, предназначенных для создания и модификации меню.

Функция API **CreateMenu** предназначена для создания главного меню приложения:

```
HMENU CreateMenu(VOID);
```

Необходимо отметить, что меню, созданное этой функцией, изначально будет пустым, т.е. не будет содержать ни одного пункта. Заполнить меню пунктами можно с помощью функций API **AppendMenu** или **InsertMenu**.

Функция API **CreatePopupMenu** предназначена для создания всплывающего меню приложения:

```
HMENU CreatePopupMenu(VOID);
```

Всплывающее меню, созданное этой функцией, также изначально будет пустым. Заполнить меню пунктами можно с помощью уже упомянутых функций API **AppendMenu** или **InsertMenu**.

Функция API **AppendMenu** применяется для добавления пунктов к концу меню.

```
BOOL AppendMenu(
    HMENU hMenu, //дескриптор меню, к которому добавляется
                 //новый пункт
    UINT uFlags, //внешний вид и правило поведения
                 //добавляемого пункта меню
    UINT_PTR uIDNewItem, //идентификатор для нового пункта
                 //меню или дескриптор выпадающего меню
    LPCTSTR lpNewItem //содержимое нового пункта меню -
                 //зависит от второго параметра
);
```

Второй параметр может иметь одно или несколько значений (флагов), приведенных ниже. В последнем случае флаги объединяются с помощью побитовой операции «ИЛИ».

- **MF_POPUP** — создает всплывающее меню. В этом случае третий параметр функции содержит дескриптор всплывающего меню.
- **MF_CHECKED** — помещает отметку рядом с пунктом меню.
- **MF_DEFAULT** — пункт меню установлен в качестве применяемого по умолчанию. Имя этого пункта выделяется жирным шрифтом. В этом случае при открытии подменю двойным щелчком мыши, Windows автоматически выполнит команду по умолчанию, закрыв при этом подменю.
- **MF_ENABLED** — делает пункт меню доступным для выбора.
- **MF_DISABLED** — делает пункт меню недоступным для выбора.
- **MF_GRAYED** — выделяет серым цветом пункт меню и запрещает его выбор.

- **MF_UNCHECKED** — пункт меню не имеет отметки.
- **MF_SEPARATOR** — указывает, что пункт меню является разделителем (сепаратором).
- **MF_STRING** — отображает пункт меню с использованием текстовой строки, которая задается в четвертом параметре.

Функция API **InsertMenu** позволяет вставить новый пункт в меню.

```
BOOL InsertMenu(
    HMENU hMenu, //дескриптор меню, которое модифицируется
    UINT uPosition, //идентификатор или позиция пункта
                    //меню, перед которым происходит
                    //вставка нового пункта

    UINT uFlags, //интерпретация второго параметра, а также
                 //внешний вид и правило поведения нового
                 //пункта меню

    PTR uIDNewItem, //идентификатор для нового пункта меню

                    //или дескриптор выпадающего меню
    LPCTSTR lpNewItem //содержимое нового пункта меню -
                      //зависит от третьего параметра
);
```

Второму параметру функции **uPosition** передается либо идентификатор пункта меню, либо позиция пункта меню. Выбор варианта интерпретации задается в третьем параметре:

- **MF_BYCOMMAND** — в этом случае второй параметр должен содержать идентификатор пункта меню, перед которым происходит вставка нового пункта;

- **MF_BYPOSITION** — в этом случае второй параметр должен содержать относительную позицию пункта меню с отсчетом от нуля.

Кроме того, в третьем параметре можно дополнительно указать один или несколько флагов, приведенных выше при рассмотрении функции **AppendMenu**.

Функция API **ModifyMenu** применяется для изменения существующего пункта меню.

```
BOOL ModifyMenu(
    HMENU hMenu, //дескриптор меню, которое модифицируется
    UINT uPosition, //идентификатор или позиция пункта
                    //меню, который будет изменен
    UINT uFlags, //интерпретация второго параметра,
                 //а также внешний вид и правило поведения
                 //изменяемого пункта меню
    PTR uIDNewItem, //идентификатор для изменяемого пункта
                    //меню или дескриптор выпадающего меню
    LPCTSTR lpNewItem //новое содержимое изменяемого пункта
                      //меню - зависит от третьего параметра
);
```

Назначение параметров функции **ModifyMenu** аналогично функции **InsertMenu**.

Функция API **DeleteMenu** применяется для удаления отдельного пункта из указанного меню.

```
BOOL DeleteMenu(
    HMENU hMenu, //дескриптор меню, которое модифицируется
    UINT uPosition, //идентификатор или позиция пункта
                    //меню, который будет удален
    UINT uFlags //интерпретация второго параметра
);
```

Следует отметить, что если удаляемым пунктом является выпадающее меню, то оно уничтожается и высвобождается память.

Важно помнить, что после вызова функций **AppendMenu**, **InsertMenu** или **DeleteMenu** с целью модификации главного меню следует обязательно вызвать рассмотренную ранее функцию API **DrawMenuBar** для повторного отображения изменившейся полосы меню.

Функция API **DestroyMenu** предназначена для уничтожения меню и высвобождения памяти, занимаемой меню.

```
BOOL DestroyMenu(
      HMENU hMenu //дескриптор уничтожаемого меню
);
```

Функция API **GetMenuString** получает строку текста указанного пункта меню.

```
int GetMenuString(

    HMENU hMenu, //дескриптор меню
    UINT uIDItem, //идентификатор или позиция пункта меню,
                  //текст которого необходимо получить
    LPTSTR lpString, //указатель на строковый буфер,
                     //в который будет записана строка текста
    int nMaxCount, //максимальное число символов, которое
                   //должно быть записано в буфер
    UINT uFlag //интерпретация второго параметра
);
```

В следующем приложении демонстрируется программный способ создания меню (исходный код приложения прикреплен к PDF-файлу урока, папка **SOURCE/ Dynamic menu**).

```
                          //header.h
#pragma once
#include <windows.h>
#include <windowsX.h>
#include <tchar.h>
#include "resource.h"

                        //MenuBarDlg.h
#pragma once
#include "header.h"

class CMenuBarDlg
{
public:
    CMenuBarDlg(void);
public:
    static BOOL CALLBACK DlgProc(HWND hWnd, UINT mes,
        WPARAM wp, LPARAM lp);
    static CMenuBarDlg* 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);
};

                        //MenuBarDlg.cpp
#include "MenuBarDlg.h"

#define ID_FILE_EXIT WM_USER
#define ID_EDIT_DELETE WM_USER+1
#define ID_EDIT_FINDNEXT WM_USER+2
#define ID_EDIT_TIMEDATE WM_USER+3
#define ID_EDIT_GOTO WM_USER+4
#define ID_VIEW_STATUSBAR WM_USER+5
#define ID_FILE_NEW WM_USER+6
```

```
#define ID_FILE_OPEN WM_USER+7
#define ID_FILE_SAVE WM_USER+8
#define ID_FILE_SAVE_AS WM_USER+9
#define ID_FILE_PAGE_SETUP WM_USER+10
#define ID_FILE_PRINT WM_USER+11
#define ID_EDIT_UNDO WM_USER+12
#define ID_EDIT_CUT WM_USER+13
#define ID_EDIT_COPY WM_USER+14
#define ID_EDIT_PASTE WM_USER+15
#define ID_EDIT_FIND WM_USER+16
#define ID_EDIT_REPLACE WM_USER+17
#define ID_EDIT_SELECT_ALL WM_USER+18

CMenuBarDlg* CMenuBarDlg::ptr = NULL;

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

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

BOOL CMenuBarDlg::Cls_OnInitDialog(HWND hwnd,
        HWND hwndFocus, LPARAM lParam)
{
    HMENU subFileMenu = CreatePopupMenu();
    AppendMenu(subFileMenu, MF_STRING, ID_FILE_NEW,
        TEXT("New"));
    AppendMenu(subFileMenu, MF_STRING, ID_FILE_OPEN,
        TEXT("Open..."));
    AppendMenu(subFileMenu, MF_STRING, ID_FILE_SAVE,
        TEXT("Save"));
    AppendMenu(subFileMenu, MF_STRING, ID_FILE_SAVE_AS,
        TEXT("Save As..."));
```

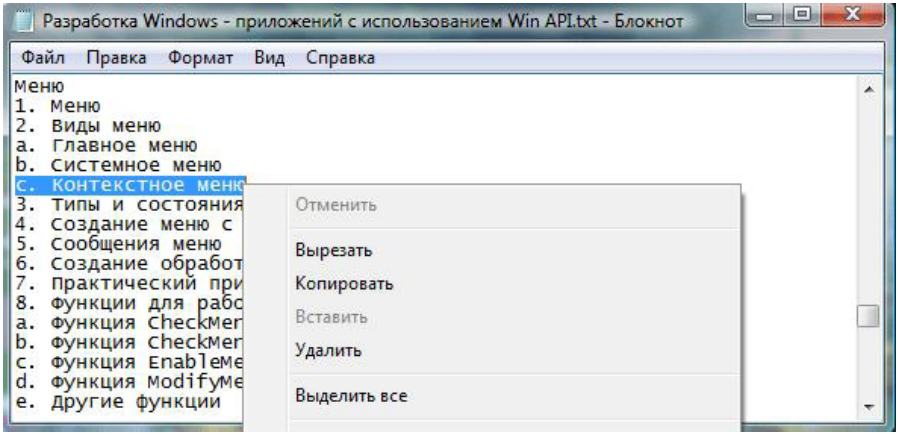
```
AppendMenu(subFileMenu, MF_SEPARATOR, 0, 0);
AppendMenu(subFileMenu, MF_STRING, ID_FILE_PAGE_SETUP,
    TEXT("Page Setup..."));
AppendMenu(subFileMenu, MF_STRING, ID_FILE_PRINT,
    TEXT("Print..."));
AppendMenu(subFileMenu, MF_SEPARATOR, 0, 0);
AppendMenu(subFileMenu, MF_STRING, ID_FILE_EXIT,
    TEXT("Exit"));
HMENU subEditMenu = CreatePopupMenu();
AppendMenu(subEditMenu, MF_STRING | MF_GRAYED,
    ID_EDIT_UNDO, TEXT("Undo"));
AppendMenu(subEditMenu, MF_SEPARATOR, 0, 0);
AppendMenu(subEditMenu, MF_STRING | MF_GRAYED,
    ID_EDIT_CUT, TEXT("Cut"));
AppendMenu(subEditMenu, MF_STRING | MF_GRAYED,
    ID_EDIT_COPY, TEXT("Copy"));
AppendMenu(subEditMenu, MF_STRING, ID_EDIT_PASTE,
    TEXT("Paste"));
AppendMenu(subEditMenu, MF_STRING | MF_GRAYED,
    ID_EDIT_DELETE, TEXT("Delete"));
AppendMenu(subEditMenu, MF_SEPARATOR, 0, 0);
AppendMenu(subEditMenu, MF_STRING, ID_EDIT_FIND,
    TEXT("Find... "));
AppendMenu(subEditMenu, MF_STRING, ID_EDIT_FINDNEXT,
    TEXT("Find Next"));
AppendMenu(subEditMenu, MF_STRING, ID_EDIT_REPLACE,
    TEXT("Replace..."));
AppendMenu(subEditMenu, MF_STRING, ID_EDIT_GOTO,
    TEXT("Go To..."));
AppendMenu(subEditMenu, MF_SEPARATOR, 0, 0);
AppendMenu(subEditMenu, MF_STRING, ID_EDIT_SELECT_ALL,
    TEXT("Select All"));
AppendMenu(subEditMenu, MF_STRING, ID_EDIT_TIMEDATE,
    TEXT("Time/Date"));
HMENU subViewMenu = CreatePopupMenu();
AppendMenu(subViewMenu, MF_STRING | MF_CHECKED,
    ID_VIEW_STATUSBAR, TEXT("Status Bar"));
```

```
    HMENU mainMenu = CreateMenu();
    AppendMenu(mainMenu, MF_POPUP,
        (UINT_PTR)subFileMenu, TEXT("File"));
    AppendMenu(mainMenu, MF_POPUP,
        (UINT_PTR)subEditMenu, TEXT("Edit"));
    AppendMenu(mainMenu, MF_POPUP,
        (UINT_PTR)subViewMenu, TEXT("View"));
    SetMenu(hwnd, mainMenu);
    return TRUE;
}

//Обработчик сообщения WM_COMMAND будет вызван при выборе
//пункта меню
void CMenuBarDlg::Cls_OnCommand(HWND hwnd, int id,
HWND hwndCtl, UINT codeNotify)
{
    HMENU hMenu = GetMenu(hwnd);
    TCHAR buf[30] = {0};
    GetMenuString(hMenu, id, buf, 30, MF_BYCOMMAND);
    MessageBox(hwnd, buf, TEXT("Динамическое создание меню"),
        MB_ICONINFORMATION | MB_OK);
}

BOOL CALLBACK CMenuBarDlg::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);
    }
    return FALSE;
}

                        //MenuBar.cpp
#include "MenuBarDlg.h"
```

```
int WINAPI _tWinMain(HINSTANCE hInst, HINSTANCE hPrev,
    LPTSTR lpszCmdLine, int nCmdShow)
{
    CMenuBarDlg dlg;
    return DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1),
        NULL, CMenuBarDlg::DlgProc);
}
```

6. Контекстное меню

Как отмечалось ранее, ***контекстное меню*** — это меню, которое появляется в любой части окна приложения при щелчке правой кнопкой мыши. При этом содержание контекстного меню изменяется в зависимости от ***«контекста»*** — места, где произведен щелчок правой кнопкой мыши. Обычно контекстное меню содержит пункты, которые дублируют команды основного меню, но сгруппированные иначе, чтобы максимально облегчить пользователю работу с приложением.

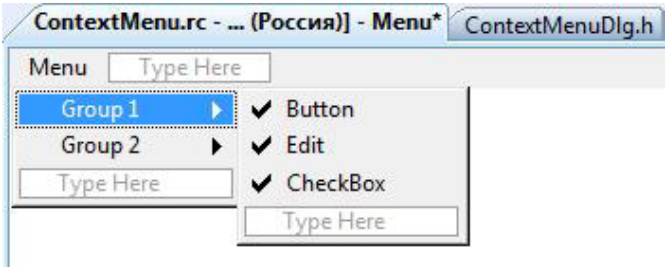


Создание контекстного меню выполняется аналогично созданию главного меню приложения. Для этого используется два подхода:

- определение шаблона меню в файле описания ресурсов;
- программное (динамическое) создание меню.

Создавая шаблон контекстного меню с помощью редактора меню, необходимо определить нулевой пункт меню

нулевого уровня как подменю, имеющее какое-нибудь условное имя. Этот пункт нигде не будет отображаться. Он необходим только для получения дескриптора подменю с помощью функции API **GetSubMenu**, рассмотренной ранее.



После определения контекстного меню в файле описания ресурсов необходимо его загрузить с помощью функции API **LoadMenu**. Данная функция вернет дескриптор фиктивного меню нулевого уровня, которое не должно отображаться на экране.

Содержанием контекстного меню является нулевой пункт указанного меню, поэтому окончательное значение дескриптора контекстного меню определяется вызовом функции **GetSubMenu**. Полученный дескриптор контекстного меню затем передается функции **TrackPopupMenu**, которая выводит всплывающее контекстное меню на экран.

```
BOOL TrackPopupMenu(
    HMENU hMenu, //дескриптор контекстного меню
    UINT uFlags, //флаги, определяющие позиционирование
                 //и другие опции меню
    int x, //горизонтальное расположение контекстного меню
           //в экранных координатах
    int y, //вертикальное расположение контекстного меню
           //в экранных координатах
```

```
    int nReserved, //зарезервированное значение; должно
                   //быть равно 0
    HWND hWnd, //дескриптор окна, которому принадлежит
               //контекстное меню
    HWND prcRect //параметр игнорируется
);
```

Описанные выше действия иллюстрируются следующим фрагментом кода:

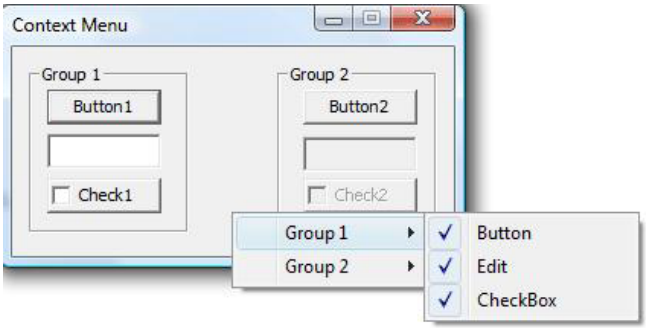
```
//Загрузим меню из ресурсов приложения
HMENU hMenu = LoadMenu(GetModuleHandle(NULL),
          MAKEINTRESOURCE(IDR_MENU1));
//Получим дескриптор подменю
HMENU hSubMenu = GetSubMenu(hMenu, 0);
//Отобразим контекстное меню, левый верхний угол которого
//привязывается к точке текущего положения курсора мыши
TrackPopupMenu(hSubMenu, TPM_LEFTALIGN, xPos, yPos, 0,
          hDialog, NULL);
```

Как известно, при щелчке правой кнопкой мыши Windows отправляет оконной процедуре приложения сообщение **WM_RBUTTONDOWN**, содержащее в **LPARAM** клиентские координаты курсора мыши в момент щелчка. Помимо этого сообщения в оконную процедуру приложения приходит сообщение **WM_CONTEXTMENU**, содержащее в **LPARAM** экранные координаты курсора мыши (в младшей части **LPARAM** находится координата X положения курсора мыши, а в старшей части — координата Y), а в **WPARAM** дескриптор окна, в котором произведен щелчок.

Учитывая, что рассмотренная выше функция отображения контекстного меню **TrackPopupMenu** принимает экранные координаты позиции расположения меню, то было бы удобно предусмотреть в приложении обработчик

сообщения **WM_CONTEXTMENU** вместо **WM_RBUTTONDOWN** для организации вызова контекстного меню. В этом случае не нужно выполнять преобразование клиентских координат в экранные координаты.

В качестве примера использования контекстного меню рассмотрим следующее приложение, в котором на форме диалога расположены две группы элементов управления. С помощью контекстного меню происходит управление состоянием элементов управления — разрешение или запрещение элементов управления (исходный код приложения прикреплен к PDF-файлу урока, папка **SOURCE/ Context menu**).




```
                        // header.h
#pragma once
#include <windows.h>
#include <windowsX.h>
#include <tchar.h>
#include "resource.h"

                    // ContextMenuDlg.h
#pragma once
#include "header.h"
```

```
class CContextMenuDlg
{
public:
    CContextMenuDlg(void);
public:
    static BOOL CALLBACK DlgProc(HWND hWnd, UINT mes,
        WPARAM wp, LPARAM lp);
    static CContextMenuDlg* 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_OnContextMenu(HWND hwnd, HWND hwndContext,
        UINT xPos, UINT yPos);
    HWND hDialog, hButton1, hButton2, hEdit1, hEdit2,
        hCheck1, hCheck2;
    HMENU hMenu, hSubMenu, hSubMenu2, hSubMenu3;
    DWORD dwButtonState, dwEditState, dwCheckboxState;
};
                    // ContextMenuDlg.cpp
#include "ContextMenuDlg.h"

CContextMenuDlg* CContextMenuDlg::ptr = NULL;
CContextMenuDlg::CContextMenuDlg(void)
{
    ptr = this;
    dwButtonState = dwEditState = dwCheckboxState = MF_CHECKED;
}

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

BOOL CContextMenuDlg::Cls_OnInitDialog(HWND hwnd,
        HWND hwndFocus, LPARAM lParam)
```

```
{
    hDialog = hwnd;
    //Получим дескрипторы элементов управления
    hEdit1 = GetDlgItem(hDialog, IDC_EDIT1);
    hEdit2 = GetDlgItem(hDialog, IDC_EDIT2);
    hButton1 = GetDlgItem(hDialog, IDC_BUTTON1);
    hButton2 = GetDlgItem(hDialog, IDC_BUTTON2);
    hCheck1 = GetDlgItem(hDialog, IDC_CHECK1);
    hCheck2 = GetDlgItem(hDialog, IDC_CHECK2);
    //Загрузим меню из ресурсов приложения
    hMenu = LoadMenu(GetModuleHandle(NULL),
        MAKEINTRESOURCE(IDR_MENU1));

    //Получим дескрипторы подменю
    hSubMenu = GetSubMenu(hMenu, 0);
    hSubMenu2 = GetSubMenu(hSubMenu, 0);
    hSubMenu3 = GetSubMenu(hSubMenu, 1);

    //Установим отметку на первом пункте в радио-группе
     //элементов меню
    CheckMenuRadioItem(hSubMenu3, ID_GROUP2_BUTTON,
        ID_GROUP2_CHECKBOX, ID_GROUP2_BUTTON, MF_BYCOMMAND);
    return TRUE;
}

//Обработчик сообщения WM_COMMAND будет вызван при выборе
//пункта меню
void CContextMenuDlg::Cls_OnCommand(HWND hwnd, int id,
        HWND hwndCtl, UINT codeNotify)
{
    if(id >= ID_GROUP1_BUTTON && id <= ID_GROUP1_CHECKBOX)
    {
    switch(id)
    {
        case ID_GROUP1_BUTTON:
            //Узнаем текущее состояние пункта меню
            if(dwButtonState == MF_CHECKED)
                EnableWindow(hButton1, FALSE);
```

```
            else
                EnableWindow(hButton1, TRUE);
                //Изменим состояние пункта меню на
                //противоположное, т.е. если пункт меню
                //был отмечен, то снимем отметку и наоборот
                dwButtonState ^= MF_CHECKED;
                CheckMenuItem(hSubMenu2, ID_GROUP1_BUTTON,
                MF_BYCOMMAND | dwButtonState);
                break;
            case ID_GROUP1_EDIT:
                if(dwEditState == MF_CHECKED)
                    EnableWindow(hEdit1, FALSE);
                else
                    EnableWindow(hEdit1, TRUE);
                    dwEditState ^= MF_CHECKED;
                    CheckMenuItem(hSubMenu2, ID_GROUP1_EDIT,
                        MF_BYCOMMAND | dwEditState);
                    break;
            case ID_GROUP1_CHECKBOX:
                if(dwCheckboxState == MF_CHECKED)
                    EnableWindow(hCheck1, FALSE);
                else
                    EnableWindow(hCheck1, TRUE);
                    dwCheckboxState ^= MF_CHECKED;
                    CheckMenuItem(hSubMenu2,
                        ID_GROUP1_CHECKBOX, MF_BYCOMMAND |
                         dwCheckboxState);
                    break;
        }
    }

    if(id >= ID_GROUP2_BUTTON && id <= ID_GROUP2_CHECKBOX)
    {
    //Установим отметку на выбранном пункте в радио-группе
     //элементов меню
    CheckMenuRadioItem(hSubMenu3, ID_GROUP2_BUTTON,
            ID_GROUP2_CHECKBOX, id, MF_BYCOMMAND);
```

```
    switch(id)
    {
        case ID_GROUP2_BUTTON:
            EnableWindow(hButton2, TRUE);
            EnableWindow(hEdit2, FALSE);
            EnableWindow(hCheck2, FALSE);
            break;
        case ID_GROUP2_EDIT:
            EnableWindow(hButton2, FALSE);
            EnableWindow(hEdit2, TRUE);
            EnableWindow(hCheck2, FALSE);
            break;
        case ID_GROUP2_CHECKBOX:
            EnableWindow(hButton2, FALSE);
            EnableWindow(hEdit2, FALSE);
            EnableWindow(hCheck2, TRUE);
            break;
    }
}
}

//Обработчик сообщения WM_CONTEXTMENU будет вызван при
//щелчке правой кнопкой мыши в любой области диалогового
//окна
void CContextMenuDlg::Cls_OnContextMenu(HWND hwnd, HWND
hwndContext, UINT xPos, UINT yPos)
{
    //Отобразим контекстное меню, левый верхний угол
    //которого привязывается к точке текущего положения
    //курсора мыши
    TrackPopupMenu(hSubMenu, TPM_LEFTALIGN, xPos, yPos, 0,
        hDialog, NULL);
}

BOOL CALLBACK CContextMenuDlg::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_CONTEXTMENU,
            ptr->Cls_OnContextMenu);
    }
    return FALSE;
}
                    // ContextMenuDlg.cpp
#include "ContextMenuDlg.h"

int WINAPI _tWinMain(HINSTANCE hInst, HINSTANCE hPrev,
        LPTSTR lpszCmdLine, int nCmdShow)
{
    CContextMenuDlg dlg;
    return DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1),
    NULL, CContextMenuDlg::DlgProc);
}
```

Анализируя данное приложение, необходимо отметить, что контекстное меню приложения содержит два подменю:

- подменю **Group 1** включает в себя **Checkbox**-элементы меню;
- подменю **Group 2** включает в себя **Radio**-элементы меню.

Важно понимать, что в подменю, содержащем **Radio**-элементы, может быть выбран (отмечен) только один пункт меню, в то время как в подменю, содержащем **Checkbox**-элементы, одновременно могут быть отмечены несколько пунктов меню.

Как отмечалось ранее, для установки или снятия отметки предназначена функция API **CheckMenuItem**.

Функция API **CheckMenuRadioItem** отмечает выбранный пункт меню и снимает отметку со всех других опций меню в указанной группе.

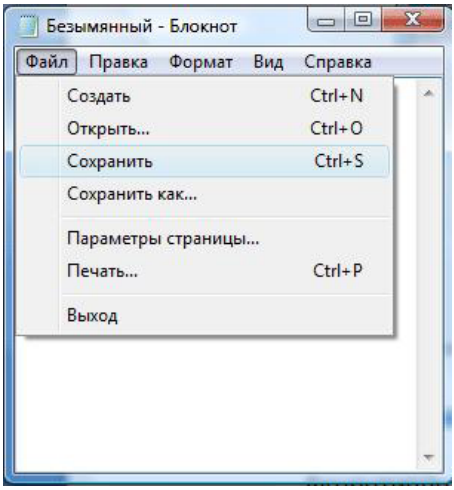
```
BOOL CheckMenuRadioItem(
    HMENU hMenu, //дескриптор меню, которое содержит группу
                 //Radio-элементов
    UINT idFirst,//первый пункт меню в группе
    UINT idLast, //последний пункт меню в группе
    UINT idCheck, //пункт меню, который должен быть отмечен
    UINT uFlags //интерпретация второго, третьего и
                //четвертого параметров
);
```

Последний параметр может принимать одно из следующих значений:

- **MF_BYCOMMAND** — в этом случае второй, третий и четвертый параметры должны содержать идентификатор пункта меню;
- **MF_BYPOSITION** — в этом случае второй, третий и четвертый параметры должны содержать относительную позицию пункта меню с отсчетом от нуля.

7. Акселераторы

Во многих Windows-приложениях (например, в приложении «Блокнот») рядом с пунктами меню часто указываются клавиатурные комбинации (например, **Ctrl+N**). Обычно эти клавиатурные комбинации (*акселераторы*) дублируют пункты меню, предоставляя альтернативный способ вызова команд меню.

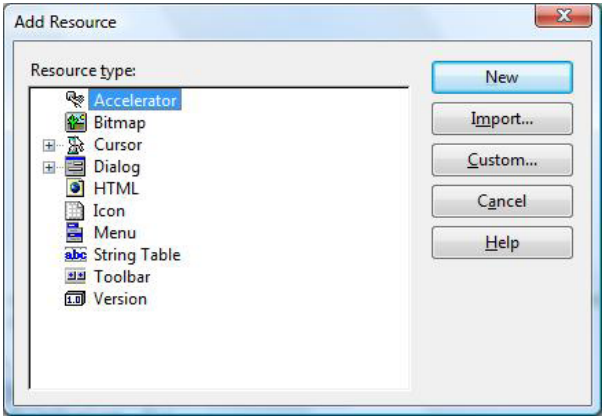


Чтобы добавить в приложение обработку акселераторов, необходимо выполнить следующую последовательность действий:

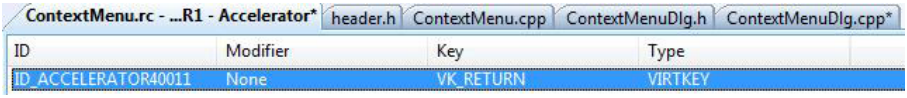
- модифицировать определение ресурса меню, добавив к имени каждого дублируемого пункта информацию о быстрой клавише;
- определить таблицу акселераторов в файле описания ресурсов;

- обеспечить загрузку таблицы акселераторов в память приложения;
- модифицировать цикл обработки сообщений в функции **WinMain**.

Для того чтобы определить таблицу акселераторов в файле описания ресурсов необходимо воспользоваться редактором таблиц акселераторов. Для этого необходимо активизировать вкладку **Resource View**, в которой с помощью контекстного меню вызвать диалог добавления ресурса **Add -> Resource...**

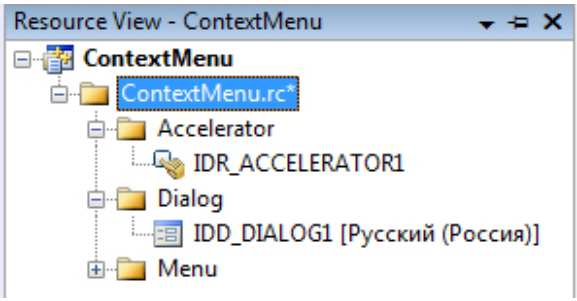


В этом диалоговом окне необходимо выбрать из списка **Accelerator** и нажать кнопку **New**, в результате чего будет открыто окно редактора таблицы акселераторов.



По умолчанию редактор присваивает таблице акселераторов имя **IDR_ACCELERATOR1**. Впоследствии этот

идентификатор можно изменить на другой идентификатор, отражающий семантику ресурса.



Поскольку акселераторы ставятся в соответствие командам меню, то они должны иметь те же идентификаторы, что и элементы меню, которым они соответствуют. Данное требование не является обязательным при реализации акселераторов, однако если оно не будет соблюдаться, то идея акселераторов как средства быстрого вызова команд меню потеряет смысл.

ID	Modifier	Key	Type
ID_CREATE	Ctrl	N	VIRTKEY
ID_FIND	Ctrl	F	VIRTKEY
ID_GO_TO	Ctrl	G	VIRTKEY
ID_OPEN	Ctrl	O	VIRTKEY
ID_QUIT	Ctrl	Q	VIRTKEY
ID_REPLACE	Ctrl	H	VIRTKEY
ID_SAVE	Ctrl	S	VIRTKEY
ID_SELECT_ALL	Ctrl	A	VIRTKEY
ID_TIME_DATE	None	VK_F5	VIRTKEY
ID_UNDO	Ctrl	Z	VIRTKEY
IDC_DECREASE	None	VK_SUBTRACT	VIRTKEY
IDC_INCREASE	None	VK_ADD	VIRTKEY

Как видно из вышеприведенного рисунка для 12 команд меню определены акселераторы. В первом столбце таблицы акселераторов указаны идентификаторы акселераторов,

совпадающие с идентификаторами пунктов меню. Во втором и третьем столбцах указаны клавиатурные комбинации. Причем клавиша, указанная в третьем столбце определена как виртуальная, о чем свидетельствует ключевое слово **VIRTKEY** в четвертом столбце (для алфавитно-цифровых клавиш и клавиш символов пунктуации используется тип ASCII).

Во время работы приложения для загрузки таблицы акселераторов в память и получения ее дескриптора используется функция API **LoadAccelerators**:

```
HACCEL LoadAccelerators(
    HINSTANCE hInstance, //дескриптор приложения
    LPCTSTR lpTableName  //указатель на строку, содержащую
                         //имя таблицы акселераторов
);
```

Для обработки акселераторов приложение должно перехватывать сообщения клавиатуры, анализировать их коды и в случае совпадения с кодом, определенным в таблице акселераторов, направлять соответствующее сообщение в оконную процедуру главного окна. С этой целью используется функция API **TranslateAccelerator**:

```
int TranslateAccelerator(
    HWND hWnd, //дескриптор окна-получателя сообщений
    HACCEL hAccTable,//дескриптор таблицы акселераторов
    LPMSG lpMsg //указатель на структуру MSG
);
```

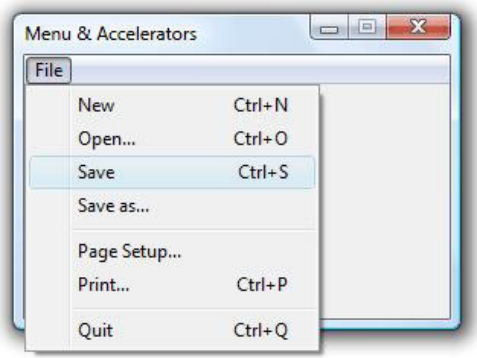
Функция **TranslateAccelerator** преобразует сообщение **WM_KEYDOWN** или **WM_SYSKEYDOWN** в сообщение **WM_COMMAND**

или **WM_SYSCOMMAND**, если таблица акселераторов содержит соответствующий код виртуальной клавиши (с учетом состояния клавиш <**Ctrl**>, <**Alt**> и <**Shift**>).

Сформированное сообщение, содержащее идентификатор акселератора в младшем слове параметра wParam, отправляется непосредственно в оконную процедуру, минуя очередь сообщений. Возврат из функции **TranslateAccelerator** происходит только после того, как оконная процедура обработает посланное сообщение.

Если функция **TranslateAccelerator** возвращает ненулевое значение, это значит, что преобразование комбинации клавиш и обработка отправленного сообщения завершились успешно. В этом случае приложение не должно повторно обрабатывать ту же самую комбинацию клавиш при помощи функций **TranslateMessage** и **DispatchMessage**.

Для демонстрации техники использования акселераторов рассмотрим следующее приложение (исходный код приложения прикреплен к PDF-файлу урока, папка **SOURCE/ Accelerators**).



```
                        // header.h
#pragma once
#include <windows.h>
#include <windowsX.h>
#include <tchar.h>
#include "resource.h"

                    // AcceleratorsDlg.h
#pragma once
#include "header.h"

class CAcceleratorsDlg
{
public:
    CAcceleratorsDlg(void);
public:
    static BOOL CALLBACK DlgProc(HWND hWnd, UINT mes,
        WPARAM wp, LPARAM lp);
    static CAcceleratorsDlg* 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);
};

                    //AcceleratorsDlg.cpp
#include "AcceleratorsDlg.h"

CAcceleratorsDlg* CAcceleratorsDlg::ptr = NULL;
CAcceleratorsDlg::CAcceleratorsDlg(void)
{
    ptr = this;
}

void CAcceleratorsDlg::Cls_OnClose(HWND hwnd)
{
```

```
    DestroyWindow(hwnd);
    PostQuitMessage(0);
}

void CAcceleratorsDlg::Cls_OnCommand(HWND hwnd, int id,
    HWND hwndCtl, UINT codeNotify)
{
    TCHAR str1[300], str2[50];
    HMENU hMenu = GetMenu(hwnd);
    GetMenuString(hMenu, id, str2, 50, MF_BYCOMMAND);
    if(codeNotify == 1)
        _tcscpy(str1, TEXT("Пункт меню выбран с помощью
                акселератора\n"));
    else if(codeNotify == 0)
        _tcscpy(str1, TEXT("Пункт меню выбран при
                непосредственном обращении к меню\n"));
        _tcscat(str1, str2);
        MessageBox(hwnd, str1, TEXT("Меню и акселераторы"),
        MB_OK | MB_ICONINFORMATION);
}

BOOL CALLBACK CAcceleratorsDlg::DlgProc(HWND hwnd,
        UINT message, WPARAM wParam, LPARAM lParam)
{
    switch(message)
    {
        HANDLE_MSG(hwnd, WM_CLOSE, ptr->Cls_OnClose);
        HANDLE_MSG(hwnd, WM_COMMAND, ptr->Cls_OnCommand);
    }
    return FALSE;
}

                    // AcceleratorsDlg.cpp
#include "AcceleratorsDlg.h"

int WINAPI _tWinMain(HINSTANCE hInst, HINSTANCE hPrev,
LPTSTR lpszCmdLine, int nCmdShow)
```

```
{
    CAcceleratorsDlg dlg;
    MSG msg;
    HWND hDialog = CreateDialog(hInst,
        MAKEINTRESOURCE(IDD_DIALOG1), NULL,
        CAcceleratorsDlg::DlgProc);
    HACCEL hAccel = LoadAccelerators(hInst,
    MAKEINTRESOURCE(IDR_ACCELERATOR1));
    ShowWindow(hDialog, nCmdShow);
    while(GetMessage(&msg, 0, 0, 0))
    {
        if(!TranslateAccelerator(hDialog, hAccel, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
    return msg.wParam;
}
```

Как видно из вышеприведенного кода можно легко отличить способ выбора пункта меню, если проанализировать четвертый параметр **codeNotify** функции — обработчика сообщения **WM_COMMAND**.

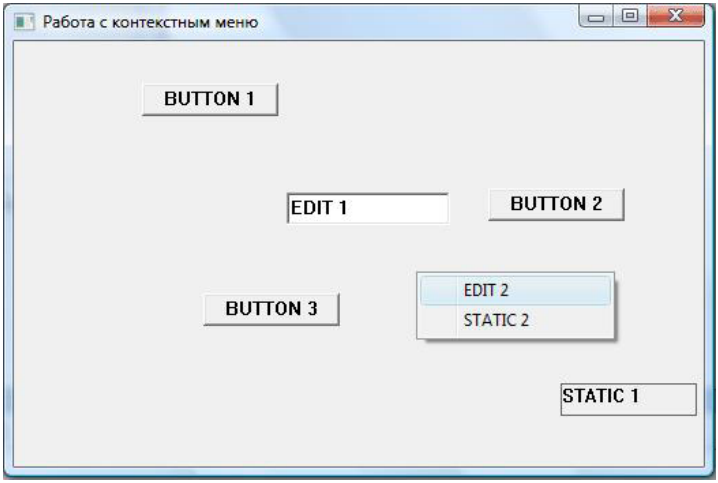
При выборе пункта меню с помощью акселератора в параметре **codeNotify** будет единичное значение, в то время как при непосредственном обращении к меню в этом параметре будет нулевое значение.

Домашнее задание

Создать в ресурсах приложения два идентичных многоуровневых меню, одно из которых содержит английские названия, а другое — русские. При старте программы необходимо загрузить англоязычное меню и присоединить его к главному окну приложения. Предусмотреть на форме диалога две кнопки. При нажатии на первую кнопку англоязычное меню должно русифицироваться, т.е. все английские названия должны быть переведены на русский язык. При повторном нажатии на эту же кнопку меню вновь должно стать англоязычным. При нажатии на вторую кнопку должны происходить аналогичные действия. Отличие состоит в том, что в первом случае одно меню должно подменяться другим, а во втором случае меню должно создаваться программным способом, а перевод меню осуществляться путем модификации каждого пункта меню. Кроме того, необходимо предусмотреть в программе возможность выбора пунктов меню с помощью акселераторов.

Разработать приложение, работа которого состоит в следующем. По щелчку правой кнопкой мыши на форме диалога выпадает контекстное меню со следующими пунктами: **«Надпись 3»**, **«Текст 3»**, **«Кнопка 3»**. Когда пользователь выбирает пункт меню, то на форме в точке щелчка появляется соответствующий элемент управления (**Static**, **Edit** или **Button**). При этом счетчик для созданного элемента управления уменьшается на единицу. Например, пользователь выбрал пункт меню **«Надпись 3»**. В этом случае

на форме в точке щелчка появится элемент управления **Static**, а при следующем отображении контекстное меню будет содержать пункты: «**Надпись 2**», «**Текст 3**», «**Кнопка 3**». Если на форме диалога выставлены все элементы одного типа, например, текст, то данный пункт исчезает из меню.



Для определения позиции курсора мыши в момент выбора пункта меню рекомендуется использовать функцию API **GetCursorPos**.

```
BOOL GetCursorPos(
      LPPOINT lpPoint //указатель на структуру POINT,
                      //в которую будут записаны экранные
                      //координаты текущей позиции курсора
                      //мыши
);
```