Модуль №4

Занятие №1

Версия 1.0.1

План занятия:

1. Повторение пройденного материала.
2. Меню. Общее понятие. Виды меню.
   2.1. Главное меню.
   2.2. Системное меню.
   2.3. Контекстное меню.
3. Типы и состояния пунктов меню.
4. Создание меню с помощью средств интегрированной среды разработки приложений Microsoft Visual Studio.
5. Обработка сообщений меню.
6. Практическая часть.
7. Подведение итогов.
8. 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) Для чего предназначена строка состояния?
2) Какие функции API позволяют создать строку состояния?
3) Чем отличается стандартный режим работы строки состояния от простого режима?

4) Какое сообщение необходимо отправить строке состояния для установки заданного режима работы?
5) С помощью какого сообщения можно разделить строку состояния на отдельные секции?
6) Каким образом можно отобразить текст в строке состояния?
7) С помощью какого сообщения можно получить клиентские координаты секции строки состояния?
8) Как установить текст всплывающей подсказки для заданной секции?
9) Как установить иконку на заданную секцию?
10) С помощью какого сообщения можно установить цвет фона строки состояния?

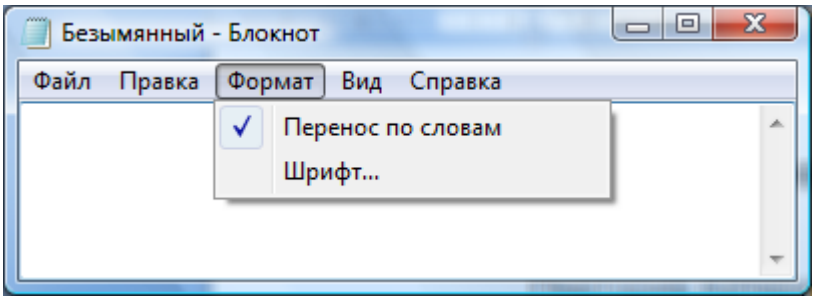
2. Меню. Общее понятие. Виды меню

Повторив материал предыдущего занятия, перейти к рассмотрению достаточно важного элемента любого приложения – **меню**. Отметить, что меню представляет собой список команд, позволяющих выполнить любую операцию, реализованную приложением. Различают следующие виды меню:

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

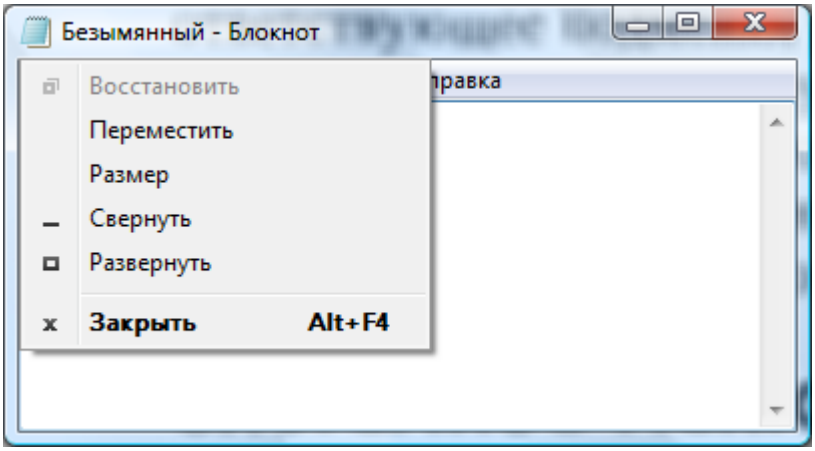
2.1. Главное меню

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.

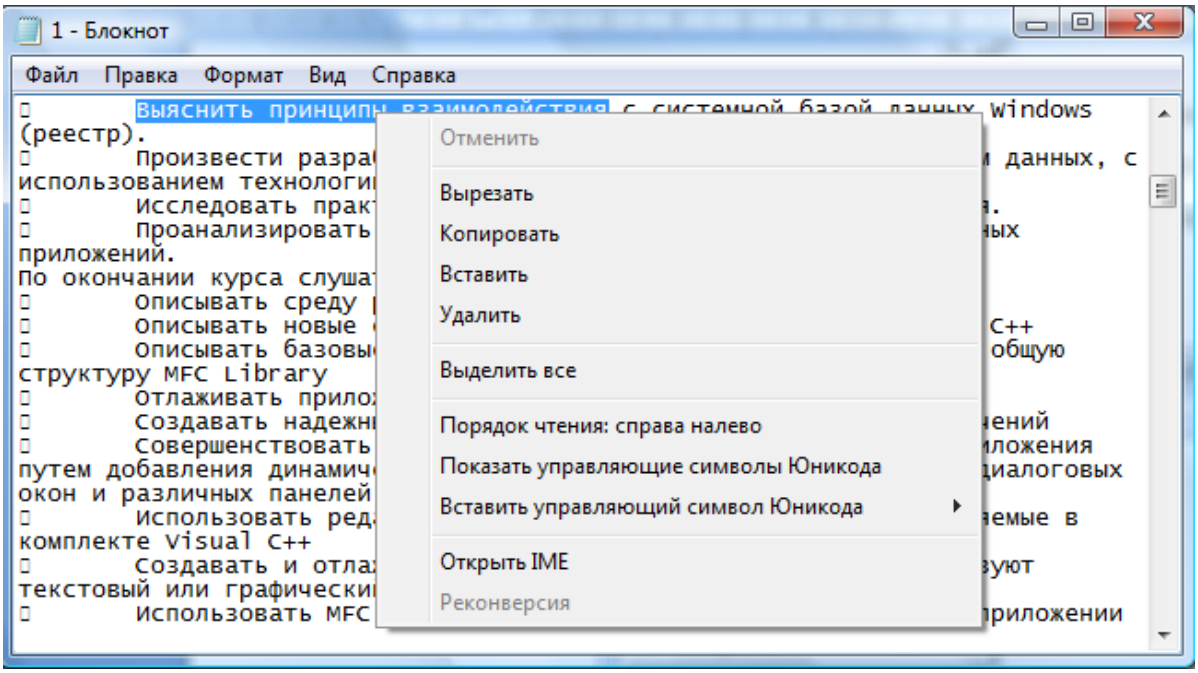
2.2. Системное меню



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.

2.3. 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.

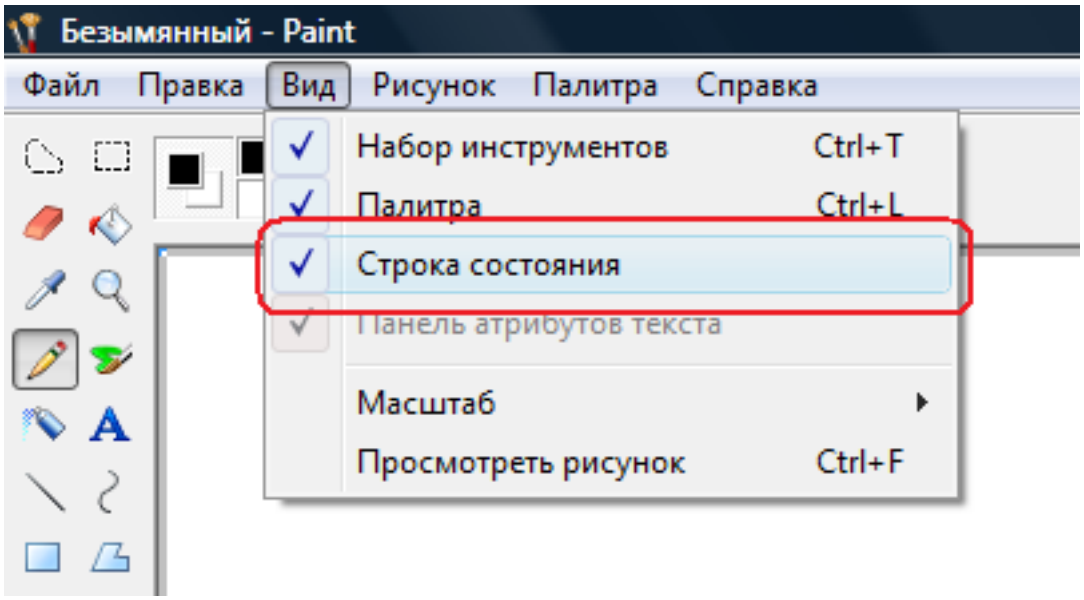


Обычно содержание контекстного меню изменяется в зависимости от «**контекста**» — места, где произведен щелчок правой кнопкой мыши. В сложных приложениях контекстные меню часто содержат пункты, дублирующие команды основного меню, но сгруппированные иначе, чтобы максимально облегчить пользователю работу с приложением. После выбора пункта контекстного меню оно исчезает с экрана.

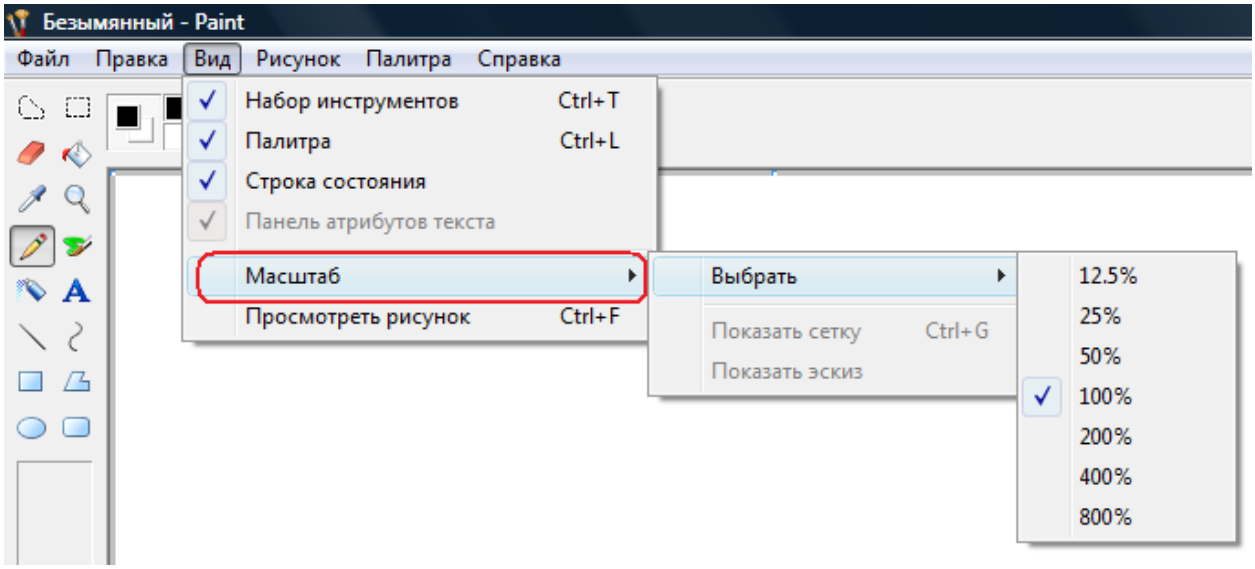
3. Типы и состояния пунктов меню

Акцентировать внимание слушателей на том, что различают два типа пунктов меню:

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

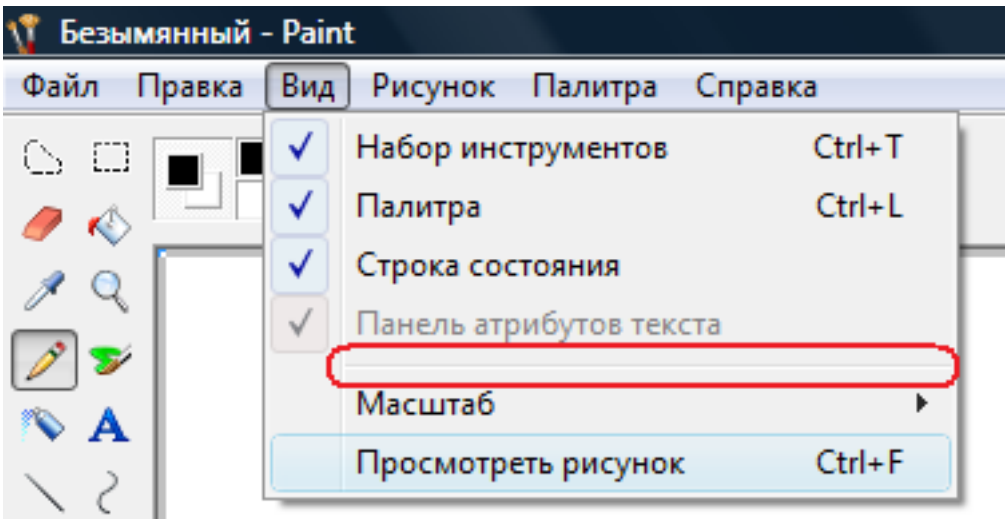


Пункт меню «**команда**» — это конечный пункт на иерархическом дереве меню. При выборе такого пункта меню обычно выполняется некоторое действие. При этом в программном коде пункту меню «команда» назначается уникальный целочисленный идентификатор.



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

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

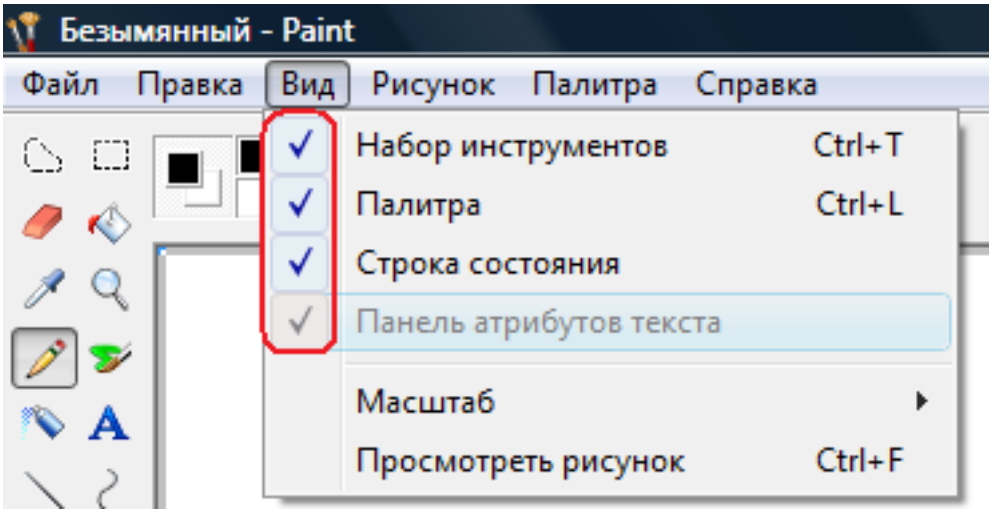


Акцентировать внимание слушателей на том, что пункты меню могут быть **разрешенными** (**enabled**), **запрещенными** (**disabled**) и **недоступными** (**grayed**). По умолчанию пункт меню является разрешенным. Когда выбирается такой пункт, система посылает оконной процедуре сообщение **WM_COMMAND**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

Запрещенный и недоступный пункты с точки 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.

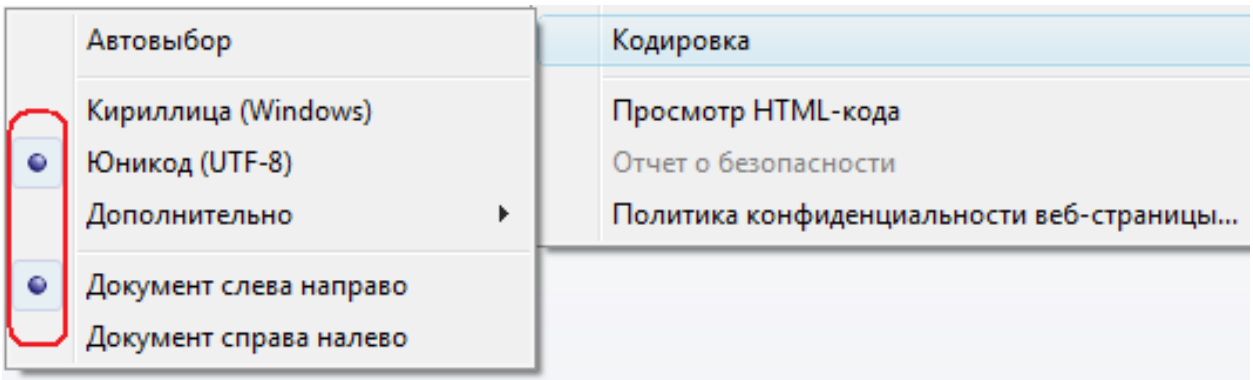
Необходимо отметить, что в хорошо продуманном интерфейсе пользователя приложение должно управлять статусом пунктов меню в зависимости от текущего состояния программы. Например, команда копирования текста в буфер обмена Windows не имеет смысла, если не вы-

делен 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.



Обратить внимание слушателей, что 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.



Следует отметить, что пункты меню могут использоваться также в роли переключателей (**radio button**). Переключатели, как и флажки,

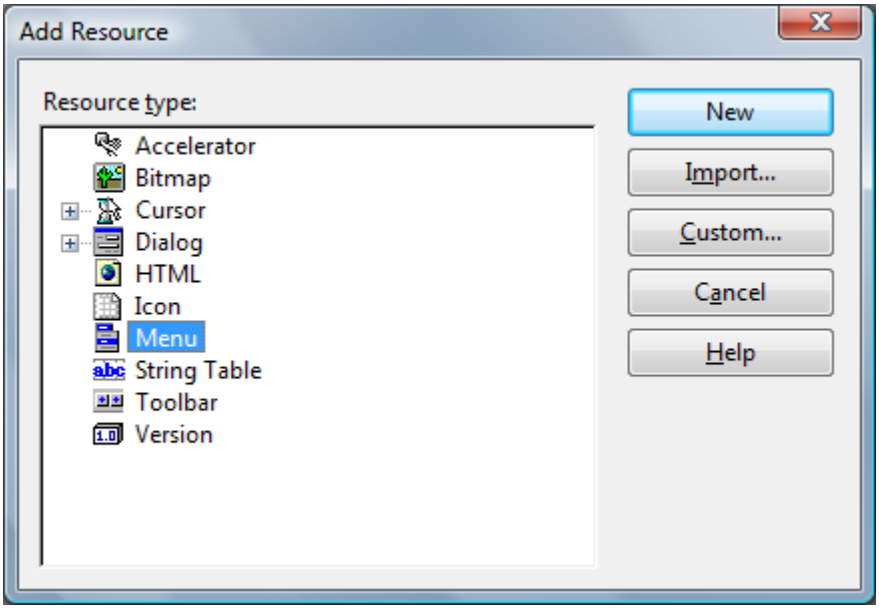
обычно используются в группе. В отличие от флажков, переключатели связываются только с взаимоисключающими опциями, поэтому в группе можно выбрать только один переключатель.

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

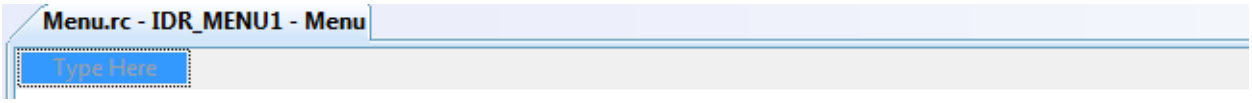
Включение меню в приложение требует следующих шагов:

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

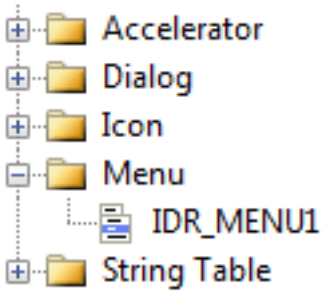
Для того чтобы определить меню в файле описания ресурсов необходимо активизировать вкладку **Resource View**, в которой с помощью контекстного меню вызвать диалог добавления ресурса **Add -> Resource...**



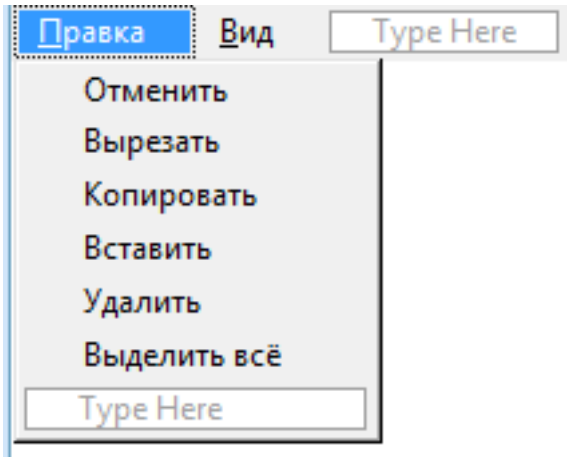
В этом диалоговом окне необходимо выбрать из списка **Menu** и нажать кнопку **New**, в результате чего будет открыто окно редактора меню с заготовкой полосы нового меню.



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



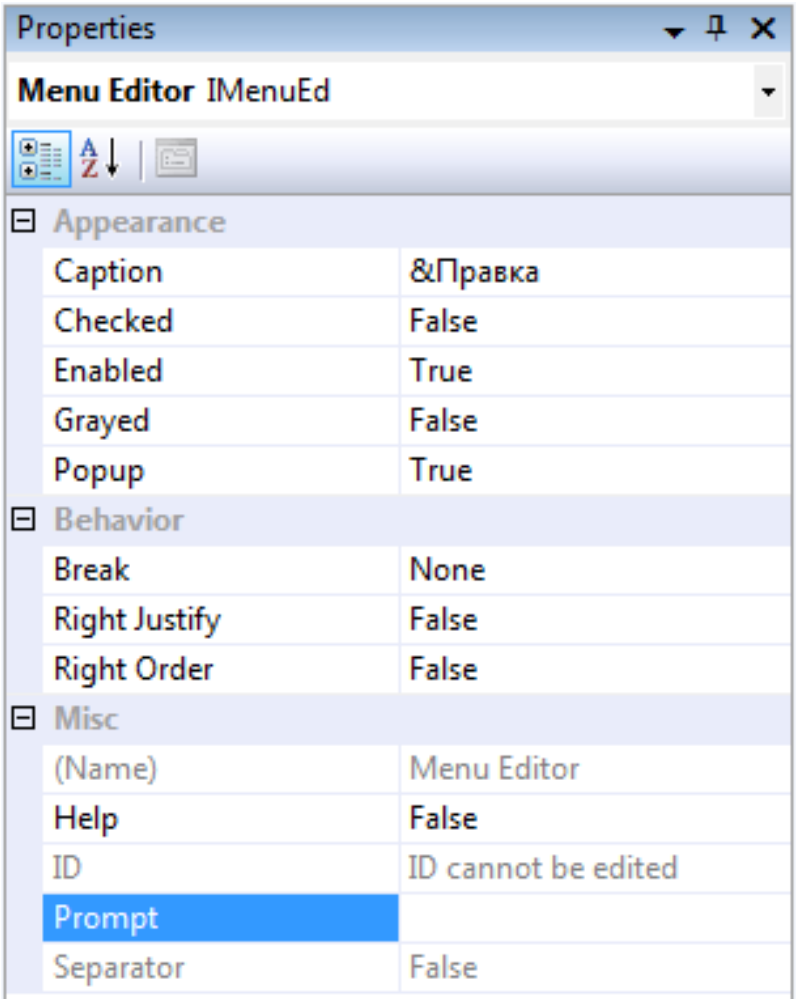
Впоследствии этот идентификатор можно изменить на другой идентификатор, отражающий семантику ресурса.



Необходимо ознакомить слушателей с некоторыми свойствами 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.

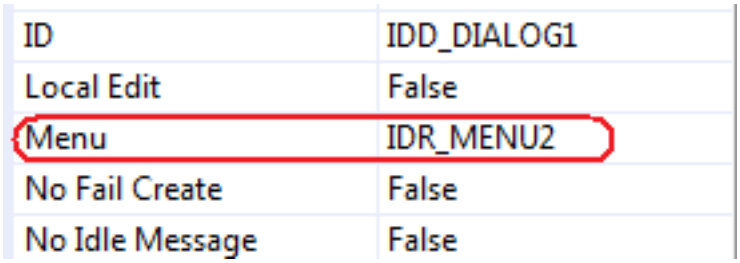


- Свойство **Grayed** обуславливает недоступность пункта меню в исходном состоянии.
- Свойство **Popup** при значении **True** определяет подменю. В противном случае пункт меню является обычной командой.
- Свойство **Break** может принимать одно из трех значений:
  - **None** — обычный пункт меню;
  - **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
);
```

5. 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 идентифи-

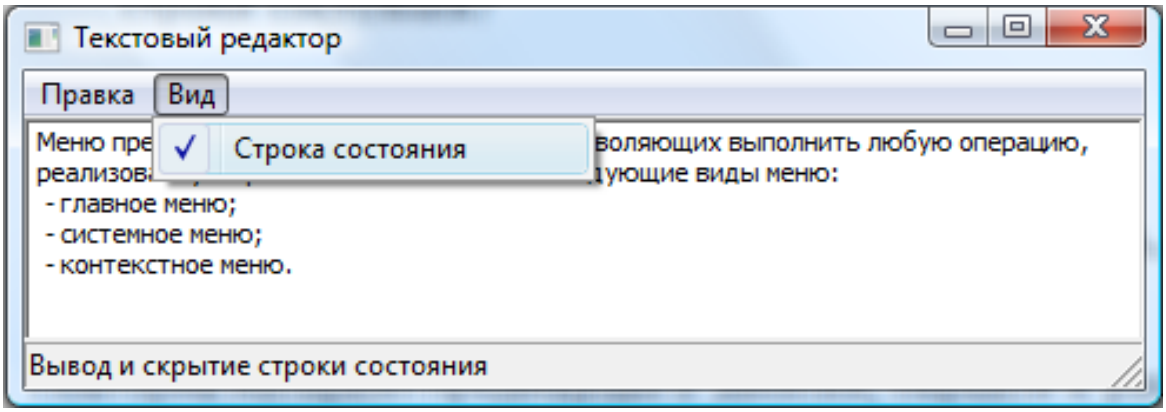
катор пункта меню, а **HIWORD(wParam)** и **lParam** содержат нулевые значения.

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 рекомендуется рассмотреть со слушателями приложение, в котором обрабатываются описанные выше сообщения меню.




```
                                    // 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 "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);
}
```

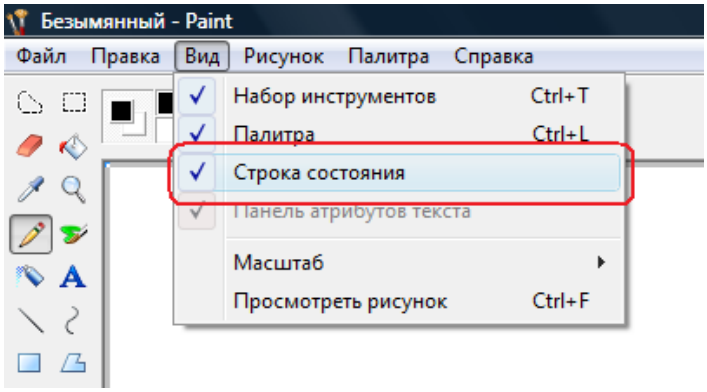
Как видно из вышеприведенного кода приложение обрабатывает сообщение **WM_INITMENUPOPUP**, которое приходит в оконную процедуру диалога непосредственно перед активизацией выпадающего меню. В обработчике этого сообщения наиболее удобно управлять статусом пунктов меню в зависимости от текущего состояния 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.

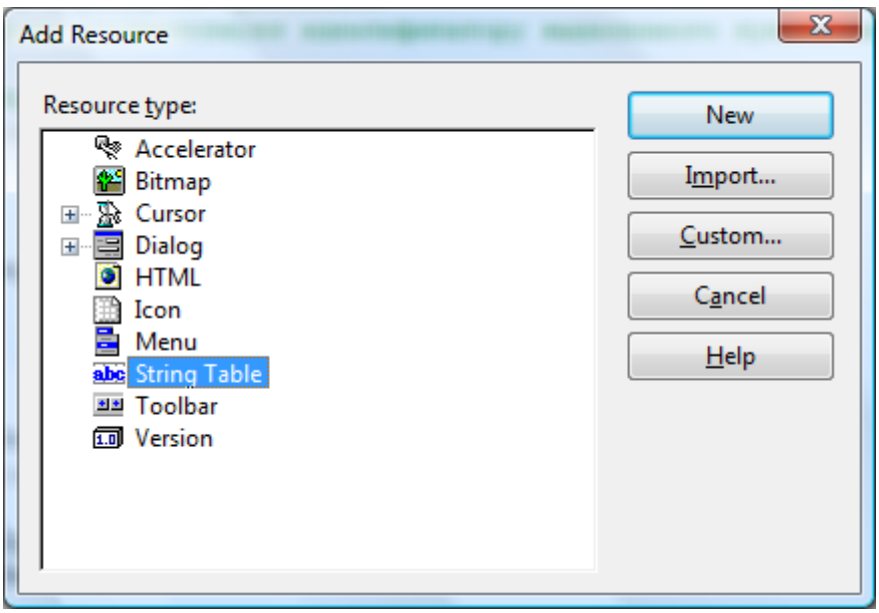
В функциях **CheckMenuItem** и **EnableMenuItem** следует рекомендовать слушателям использование флага **MF_BYCOMMAND**, передавая второму параметру идентификатор пункта меню. Это предотвратит возможные проблемы в процессе сопровождения программы, если по-

требуется 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.

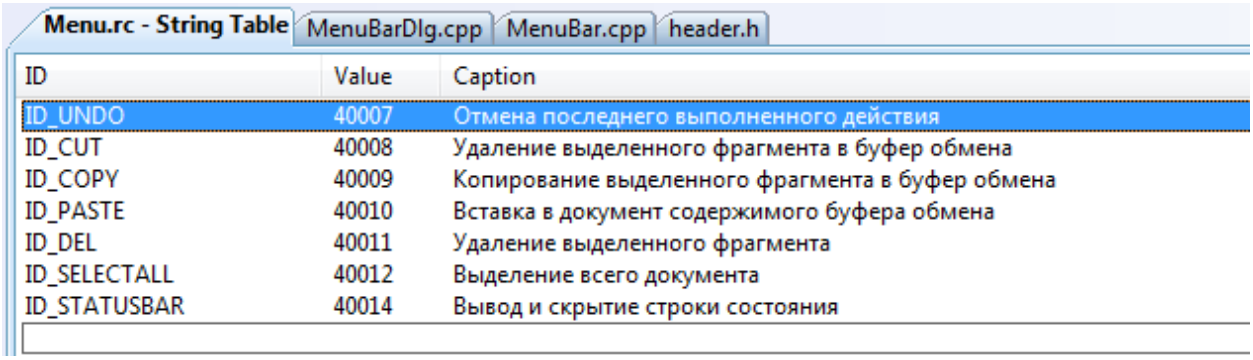
Продолжая анализировать вышеприведенный код, акцентировать внимание слушателей на обработке сообщения **WM_MENUSELECT**. Как отмечалось ранее, это сообщение удобно использовать для формирования контекстной справки меню, которая отображается в строке состояния приложения. В данном приложении для отображения 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.



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
);
```

6. Практическая часть

Разработать приложение с таким же главным меню, как и в приложении «Блокнот». В дальнейшем эта заготовка будет использоваться слушателями при написании своего собственного «Блокнота».

7. Подведение итогов

Подвести общие итоги занятия. Ещё раз подчеркнуть три основных вида меню. Выделить возможные типы и состояния пунктов меню. Перечислить сообщения, наиболее часто обрабатываемые при использовании меню. Отметить удобство использования ресурса таблицы строк. Акцентировать внимание слушателей на наиболее тонких моментах изложенной темы.

8. 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 кнопку, при нажатии на которую англоязычное меню должно русифицироваться, т. е. все английские названия должны быть переведены на русский язык. При повторном нажатии на кнопку меню вновь должно стать англоязычным.