**ЛАБОРАТОРНАЯ РАБОТА № 2**

**ОКОННЫЕ ПРИЛОЖЕНИЯ WINDOWS**

**Цель работы**

Получение практических навыков создания оконных приложений Windows на языке C/C++ с применением Win32 API.

**Основные понятия**

Основой приложений Windows являются *окна*, взаимодействие между которыми осуществляется через *сообщения*. Сообщение являет ся уведомлением о том, что произошло некоторое событие, которое может требовать, а может и не требовать выполнения определенных действий. Это событие может быть следствием действий пользователя, например, перемещение курсора или изменение размеров окна. Кроме того, событие может генерироваться приложением или же самой опе

рационной системой.

**Окна и элементы управления**

Окно в Windows – это прямоугольная область экрана, где прило жение отображает выводимую и принимает вводимую информацию от пользователя. В один и тот же момент времени только одно окно мо жет принимать данные от пользователя.

При запуске Windows автоматически создается *окно рабочего стола* (desktop window), которое окрашивает фон экрана и является владельцем всех окон, создаваемых приложениями.

Каждое приложение создает, по крайней мере, одно окно, называ емое *главным окном* (main window), которое служит основным окном программы. Главное окно относится к так называемым *перекрываю щим окнам* (overlapped windows), которые способны перекрывать окна

других приложений. Перекрывающие окна обычно включают область заголовка (title bar) и рамку (frame). Дополнительно такие окна могут иметь (а могут и не иметь) область меню (menu bar), системное меню (system menu) и кнопки управления окном: кнопка свертывания окна (minimize button), кнопка развертывания окна (maximize button), кноп ка восстановления окна (restore button) и кнопка закрытия окна (close button).

В дополнение к главному окну приложения также используют диалоговые окна и элементы управления. *Диалоговые окна*, или *окна диалога* (dialog box), – это *всплывающие окна* (pop-up windows), кото-

рые используются для получения от пользователя дополнительной информации, а также вывода результатов работы приложения. Всплы вающие окна могут иметь область заголовка и рамку окна.

*Элементами управления* (controls) являются *дочерние окна* (child windows), предназначенные для отображения и редактирования раз личной информации, а также для выполнения пользователем опреде ленных действий.

Каждое окно содержит *клиентскую область* (client area), в которой можно выводить текст и графику. Окно также может иметь (а может и не иметь) *не клиентскую область* (nonclient area), которая включает в себя область заголовка, область меню и рамку окна. Управлением кли

ентской областью окна занимается приложение, не клиентской обла стью – операционная система.

**Системы координат**

Прежде чем перейти к дальнейшему рассмотрению, следует отме тить, что в функциях Win32 API, работающих с окнами, может исполь зоваться одна из следующих систем координат:

− экранные координаты (screen coordinates);

− координаты клиентской области (client coordinates).

На рис. 1.1 показано главное окно программы на экране, а также взаимоотношение между системами координат.

(0, 0)

Заголовок окна (0, 0)

y*client*

Клиентская область

x*client*

x*screen*

y*screen*

*Рис. 1.1.* Системы координат Windows

Для преобразования координат клиентской области окна в экран ные координаты в Win32 API есть функция ClientToScreen:

BOOL **ClientToScreen**(HWND *hWnd*, LPPOINT *lpPoint*);

Здесь параметр *hWnd* содержит дескриптор окна, а параметр *lpPoint* указывает на переменную типа POINT, в которой хранятся ко ординаты клиентской области. В случае успеха функция возвращает значение отличное от FALSE и записывает результат в переменную, на которую указывает *lpPoint*.

Обратное преобразование выполняет функция ScreenToClient: BOOL **ScreenToClient**(HWND *hWnd*, LPPOINT *lpPoint*);

Она очень похожа на ClientToScreen. Здесь параметр *hWnd* также содержит дескриптор окна, а параметр *lpPoint* указывает на перемен ную типа POINT, в которой хранятся экранные координаты. В случае успеха функция возвращает значение отличное от FALSE и записывает результат в переменную, на которую указывает *lpPoint*.

**Оконный класс**

Любое окно в Windows создается на основе оконного класса. *Оконный класс* (window class) – это структура, определяющая основ ные характеристики окна. К ним относятся стиль класса и связанные с ним ресурсы, такие как меню (menu), пиктограмма (icon), курсор (cur sor) и кисть (brush) для закрашивания фона окна. Кроме того, эта структура содержит адрес функции, предназначенной для обработки сообщений любого окна данного класса. (Не следует путать понятие *оконного класса Windows* с понятием *класса C++*.)

Для создания нового оконного класса необходимо заполнить структуру WNDCLASSEX, а затем передать адрес этой структуры в виде аргумента функции RegisterClassEx.

ATOM **RegisterClassEx**(CONST WNDCLASSEX \**lpWndClassEx*);

Функция RegisterClassEx это расширенная версия функции RegisterClass из предыдущих версий Windows. Можно пользоваться и функцией RegisterClass, передавая ей адрес структуры WNDCLASS. В случае успеха функции RegisterClassEx и RegisterClass возвращают не нулевое значение.

Структура WNDCLASSEX определена следующим образом:

typedef struct tagWNDCLASSEX {

UINT *cbSize*; // *размер структуры в байтах*

UINT *style*; // *стиль оконного класса*

WNDPROC *lpfnWndProc*; // *указатель на оконную процедуру*

int *cbClsExtra*; // *число дополнительных байтов,*

// *которые должны быть распределены*

// *в конце структуры*

int *cbWndExtra*; // *число дополнительных байтов,*

// *которые должны быть распределены*

// *вслед за экземпляром окна*

HINSTANCE *hInstance*; // *дескриптор экземпляра приложения,* // *в котором находится оконная*

// *процедура для этого класса*

HICON *hIcon*; // *дескриптор пиктограммы*

HCURSOR *hCursor*; // *дескриптор курсора*

HBRUSH *hbrBackground*; // *дескриптор кисти, используемой* // *для закраски фона окна*

LPCTSTR *lpszMenuName*; // *указатель на строку, содержащую* // *имя меню, применяемого*

// *для этого класса*

LPCTSTR *lpszClassName*; // *указатель на строку, содержащую* // *имя этого класса*

HICON *hIconSm*; // *дескриптор малой пиктограммы*

} WNDCLASSEX;

Первое поле, *cbSize*, должно содержать значение равное размеру структуры в байтах, как правило, его получают с помощью оператора sizeof.

Второе поле, *style*, может содержать один или несколько стилей, объединенных с помощью операции поразрядного «ИЛИ» языка C/C++. Наиболее употребительные стили оконного класса перечис ленных в табл. 1.1. Полный список таких стилей см. в документации Platform SDK.

*Таблица 1.1.* Некоторые стили оконного класса

| **Стиль** | **Описание** |
| --- | --- |
| CS\_DBLCLKS | Оконная процедура будет получать сообщения при двой ном щелчке клавишей мыши (double click) |
| CS\_GLOBALCLASS | Создается оконный класс, доступный всем приложениям. Другие приложения могут создавать окна этого класса |
| CS\_HREDRAW | Клиентская область окна этого класса будет перерисовы ваться при изменении ширины окна |
| CS\_NOCLOSE | Запретить команду «Закрыть» в системном меню |
| CS\_OWNDC | Выделить уникальный контекст устройства для каждого окна, созданного при помощи этого класса |
| CS\_VREDRAW | Клиентская область окна этого класса будет перерисовы ваться при изменении высоты окна |

Третье поле, *lpfnWndProc*, содержит адрес оконной процедуры – функции, предназначенной для обработки сообщений любого окна данного класса (подробнее см. в разделе «Оконные сообщения»).

Поля *cbClsExtra* и *cbWndExtra* используются крайне редко. По дробнее см. в документации Platform SDK.

Шестое поле, *hInstance*, содержит дескриптор экземпляра прило жения, в котором находится оконная процедура для оконного класса. Седьмое поле *hIcon* содержит дескриптор пиктограммы, которая появляется на панели задач Windows. Значение *hIcon* обычно получа ют с помощью функции LoadIcon:

HICON **LoadIcon**(HINSTANCE *hInstance*, LPCTSTR *lpIconName*);

Эта функция загружает указанный ресурс пиктограммы, заданный параметром *lpIconName*, из экземпляра приложения, на который указы вает параметр *hInstance*. Чтобы загрузить одну из системных (пред определенных) пиктограмм, аргумент *hInstance* должен быть NULL, а параметр *lpIconName* должен содержать константу, идентификатор которой начинается с префикса IDI\_ (например, IDI\_APPLICATION).

Восьмое Поле *hCursor* содержит дескриптор курсора мыши, ис пользуемого приложением в клиентской области окна. Значение *hCursor* обычно получают вызовом функции LoadCursor:

HCURSOR **LoadCursor**(HINSTANCE *hInstance*, LPCTSTR *lpCursorName*);

Эта функция загружает ресурс курсора, на который указывает па раметр *lpCursorName*, из экземпляра приложения, заданного парамет ром *hInstance*. Чтобы загрузить один из системных (предопределен ных) курсоров, аргумент *hInstance* должен быть NULL, а параметр *lpCursorName* должен содержать константу, идентификатор которой начинается с префикса IDC\_ (например, IDC\_ARROW).

Девятое поле, *hbrBackground*, содержит дескриптор кисти, которая должна использоваться для закрашивания фона клиентской области окна. На самом деле, значение этого поля может быть, как дескрипто ром кисти, так и кодом цвета. Код цвета задается константой одного из системных цветов, к которой добавляться 1. Код цвета обязательно должен приводиться к типу HBRUSH. Константы системных цветов начинаются с префикса COLOR\_ (например, COLOR\_WINDOW, которая соот ветствует цвету фона окна).

Если значение поля *hbrBackground* равно NULL, приложение должно обрабатывать сообщение WM\_ERASEBKGND самостоятельно закрашивая фон клиентской области окна.

Десятое поле, *lpszMenuName*, указывает на строку, содержащую имя ресурса меню, применяемого для всех окон данного класса. Если для

идентификации ресурса меню используется целое число, нужно ис пользовать макрос MAKEINTRESOURCE. Если значение этого поля равно NULL, окна этого класса по умолчанию не имеют никакого меню.

Поле *lpszClassName* указывает на строку, содержащую имя окон ного класса. Имена оконных классов должны быть уникальны только в пределах одного запущенного приложения.

Последнее поле, *hIconSm*, содержит дескриптор маленькой пикто граммы, которая отображается в области заголовка окна. Значение поля *hIconSm* получают также как и значение поля *hIcon* – с помощью функции LoadIcon. Кроме того, если поле *hIconSm* установлено в NULL, система ищет ресурс пиктограммы, указанный в поле *hIcon*.

В листинге 1.1 приводится пример регистрации нового оконного класса «MyWindowClass».

*Листинг 1.1.* Пример регистрации оконного класса

1 WNDCLASSEX wcex = { sizeof(WNDCLASSEX) };

2

3 wcex.style = CS\_HREDRAW|CS\_VREDRAW|CS\_DBLCLKS;

4 wcex.lpfnWndProc = WindowProc;

5 wcex.hInstance = hInstance;

6 wcex.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

7 wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);

8 wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

9 wcex.lpszMenuName = NULL;

10 wcex.**lpszClassName** = TEXT("MyWindowClass");

11 wcex.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);

12

13 if (0 != RegisterClassEx(&wcex))

14 {

15 /\* *оконный класс успешно зарегистрирован!* \*/

16 } // *if*

Оконный класс удаляется, когда приложение, в котором он был зарегистрирован, завершается. Приложение может также использовать функцию UnregisterClass, чтобы удалить оконный класс и освободить ресурсы, связанные с ним.

BOOL **UnregisterClass**(LPCTSTR *lpClassName*, HINSTANCE *hInstance*);

Первый параметр, *lpClassName*, указывает на строку, содержащую имя оконного класса, а второй параметр, *hInstance*, содержит дескрип тор экземпляра приложения, в котором находится оконная процедура для оконного класса.

В случае неудачи функция UnregisterClass возвращает FALSE.

***Предопределенные оконные классы***

В Win32 API имеются предопределенные оконные классы *элемен тов управления общего пользования* (common controls). Большинство из них реализовано в библиотеке ComCtl32.dll. В табл. 1.2 перечислены элементы управления общего пользования и соответствующие им оконные классы.

*Таблица 1.2.* Элементы управления общего пользования

| **Элемент управления** | **Оконный класс** | **Префикс**  **флага стиля** |
| --- | --- | --- |
| Анимационное изображение (Animate) | SysAnimate32 | ACS\_ |
| Выпадающий список (Combo box) | ComboBox  ComboBoxEx32 | CBS\_  CBES\_ |
| Гиперссылка (SysLink) | SysLink | LWS\_ |
| Групповая рамка (Group box) | Button | BS\_ |
| Дерево просмотра (Tree view) | SysTreeView32 | TVS\_ |
| Заголовок списка просмотра (Header) | SysHeader32 | HDS\_ |
| Закладки (Tab control) | SysTabControl32 | TCS\_ |
| Индикатор процесса (Progress bar) | msctls\_progress32 | PBS\_ |
| Календарь (Month calendar) | SysMonthCal32 | MCS\_ |
| Кнопка (Button) | Button | BS\_ |
| Кнопка со стрелкой раскрывающегося списка (Split button) | Button | BS\_ |
| Надпись (Static text) | Static | SS\_ |
| Окно подсказки (Tooltip) | tooltips\_class32 | TTS\_ |
| Панель инструментов (Toolbar) | ToolbarWindow32 | TBSTYLE\_ |
| Переключатель (Radio button) | Button | BS\_ |
| Поле ввода IP-адреса (IP address) | SysIPAddress32 |  |
| Поле ввода горячей клавиши (Hot key) | msctls\_hotkey32 |  |
| Поле ввода даты или времени  (Data time picker) | SysDateTimePick32 | DTS\_ |
| Полоса прокрутки (Scroll bar) | ScrollBar | SBS\_ |
| Регулятор (Slider) | msctls\_trackbar32 | TBS\_ |
| Редактируемое поле (Edit box) | Edit | ES\_ |
| Список (List box) | ListBox | LBS\_ |
| Список просмотра (List view) | SysListView32 | LVS\_ |
| Строка состояния (Status bar) | msctls\_statusbar32 | SBARS\_ |

*Окончание табл. 1.2.*

| **Элемент управления** | **Оконный класс** | **Префикс**  **флага стиля** |
| --- | --- | --- |
| Счетчик или стрелки (Spin) | msctls\_updown32 | UDS\_ |
| Текстовый редактор (Rich edit) | RichEdit  RichEdit20A  RichEdit20W  RichEdit50W | ES\_ |
| Флажок (Check box) | Button | BS\_ |

Для полноты картины в табл. 1.3 перечислены еще несколько предопределенных оконных классов, которые доступны только опера ционной системе и окна этих классов нельзя создать в приложении.

*Таблица 1.3.* Системные оконные классы

| **Оконный класс** | **Описание** |
| --- | --- |
| ComboLBox | Окно списка, содержащегося в элементе управления «выпадающий список» (combo box) |
| #32768 | Окно меню |
| #32769 | Окно рабочего стола (desktop window) |
| #32770 | Диалоговое окно |
| #32771 | Окно переключения задач (task switch window) |

В документации Platform SDK сказано, чтобы использовать окон ные классы элементов управления общего пользования (кроме классов RichEdit\*) следует вызвать функцию InitCommonControlsEx, которая регистрирует оконные классы этих элементов управления. Эта функ ция определена в заголовочном файле CommCtrl.h и реализована в библиотеке ComCtl32.dll.

BOOL **InitCommonControlsEx**(LPINITCOMMONCONTROLSEX *lpInitCtrls*);

Параметр *lpInitCtrls* передает адрес структуры INITCOMMON CONTROLSEX, содержащей информацию о том, какие классы элементов управления должны быть зарегистрированы. Однако, несмотря на этот параметр, функция InitCommonControlsEx регистрирует все доступные оконные классы.

Если загрузить библиотеку ComCtl32.dll при помощи вызова функции LoadLibrary:

LoadLibrary(TEXT("ComCtl32.dll"));

то функция DllMain этой библиотеки регистрирует оконные классы всех реализованных в ней элементов управления. Следовательно, нет необходимости вызывать еще и функцию InitCommonControlsEx.

В Windows Vista и более новых версиях, библиотека ComCtl32.dll автоматически загружается при первом создании элемента управления из этой библиотеки.

Чтобы использовать элемент управления «текстовый редактор» (rich edit) также нужно загрузить DLL при помощи вызова функции LoadLibrary. В Windows имеется несколько версий текстового редак тора, каждый из которых реализован в разных библиотеках DLL. В

табл. 1.4 показано, какие библиотеки DLL соответствуют какой версии текстового редактора.

*Таблица 1.4.* Библиотеки DLL для разных версий Rich edit

| **Версия** | **DLL** | **Оконный класс** |
| --- | --- | --- |
| 1.0 | Riched32.dll | RichEdit |
| 2.0 | Riched20.dll | RichEdit20A, RichEdit20W |
| 3.0 | Riched20.dll | RichEdit20A, RichEdit20W |
| 4.1 | Msftedit.dll | RichEdit50W |

Следует обратить внимание на то, что от версии 2.0 к версии 3.0 оконный класс и имя файла DLL не изменились. Это позволяет прило жениям, изначально созданным для использования текстового редак тора версии 2.0, использовать редактор версии 3.0, не изменяя про граммный код.

Таким образом, для работы с текстовым редактором, например, версии 4.1 нужно подключить заголовочный файл Richedit.h и загру зить библиотеку Msftedit.dll:

LoadLibrary(TEXT("Msftedit.dll"));

**Создание окна**

Для создания окна определенного оконного класса вызывается функция CreateWindowEx. Функция CreateWindowEx это расширенная версия функции CreateWindow, которую тоже можно использовать.

HWND **CreateWindowEx**(DWORD *dwExStyle*, LPCTSTR *lpClassName*, LPCTSTR *lpWindowName*, DWORD *dwStyle*, int *X*, int *Y*, int *nWidth*, int *nHeight*, HWND *hWndParent*, HMENU *hMenu*, HINSTANCE *hInstance*, LPVOID *lpParam*);

Первый параметр, *dwExStyle*, задает расширенный стиль окна, применяемый совместно со стилем, определенным в параметре

*dwStyle*. Например, в качестве расширенного стиля можно задать один или несколько флагов, определенных в табл. 1.5. Полный список рас ширенных стилей можно найти в документации Platform SDK.

*Таблица 1.5.* Расширенные стили окна

| **Стиль** | **Описание** |
| --- | --- |
| WS\_EX\_ACCEPTFILES | Создается окно, способное принимать перетаскиваемые файлы методом drag-and-drop |
| WS\_EX\_CLIENTEDGE | Клиентская область создаваемого окна будет несколько углублена |
| WS\_EX\_CONTEXTHELP | Создаваемое окно включает кнопку вопросительного знака в область заголовка окна. (Этот стиль не может использоваться с WS\_MINIMIZEBOX и WS\_MAXIMIZEBOX.) При нажатии на эту кнопку курсор принимает форму вопросительного знака, и если затем пользователь вы  полняет нажатие над областью дочернего окна, то окно получает сообщение WM\_HELP |
| WS\_EX\_STATICEDGE | Создается окно с объемным бордюром (используется для элементов управления, не принимающих ввод поль зователя) |
| WS\_EX\_TOPMOST | Создается окно, которое будет помещено поверх других окон, даже если окно неактивно |
| WS\_EX\_WINDOWEDGE | Создается окно с выпуклой границей за счет некоторого углубления клиентской области |

Параметр *lpClassName* – указатель на строку, содержащую допу стимое имя оконного класса, на основе которого создается окно. Таким именем может быть имя оконного класса, зарегистрированного при помощи функции RegisterClassEx или RegisterClass, либо имя пред определенного оконного класса (см. табл. 1.2).

Третий параметр, *lpWindowName*, – указатель на строку, содержа щую имя окна. Место отображения этого имени зависит от вида окна. Например, для главного окна приложения имя выводится как заголо вок окна, а для окна класса Edit размещается в клиентской области.

Параметр *dwStyle* задает стиль окна, который может состоять из значений, указанных в табл. 1.6. Для элементов управления общего пользования дополнительно могут быть добавлены стили, идентифи каторы которых начинаются с префикса, указанного в табл. 1.2. Пол ный список стилей можно найти в документации Platform SDK.

Параметры *X* и *Y* определяют позицию верхнего левого угла окна. Для перекрывающего окна эти параметры определяются в экранных координатах, а для дочерних окон – в клиентских координатах роди-

тельского окна. Если позиция окна не важна, то можно установить значение параметра *X* равное CW\_USEDEFAULT. В этом случае операцион ная система сама определяет положение окна. Если параметр *X* имеет значение CW\_USEDEFAULT, то значение *Y* игнорируется.

*Таблица 1.6.* Стили окна

| **Стиль** | **Описание** |
| --- | --- |
| WS\_BORDER | Создается окно с тонкой рамкой |
| WS\_CAPTION | Создается окно с областью заголовка |
| WS\_CHILD | Создается дочернее окно (не допускается совместное использование с WS\_POPUP) |
| WS\_CLIPCHILDREN | В создаваемом окне клиентская область, занятая дочер ним окном, исключается при перерисовке |
| WS\_CLIPSIBLINGS | При перерисовке создаваемого дочернего окна все остальные дочерние окна, накрадывающиеся на кли ентскую область окна, будут отсечены при перерисовке |
| WS\_HSCROLL | Создается окно с горизонтальной полосой прокрутки |
| WS\_MAXIMIZEBOX | Создается окно с кнопкой развертывания |
| WS\_MINIMIZEBOX | Создается окно с кнопкой свертывания |
| WS\_OVERLAPPED | Создается перекрывающее окно, которое будет иметь область заголовка и рамку |
| WS\_OVERLAPPEDWINDOW | Сочетание стилей WS\_OVERLAPPED, WS\_CAPTION, WS\_SYSMENU, WS\_THICKFRAME, WS\_MINIMIZEBOX и WS\_MAXIMIZEBOX |
| WS\_POPUP | Создается всплывающее окно (не допускается совмест ное использование с WS\_CHILD) |
| WS\_POPUPWINDOW | Сочетание стилей WS\_BORDER, WS\_POPUP и WS\_SYSMENU. Чтобы сделать системное меню видимым, необходимо добавить стиль WS\_CAPTION |
| WS\_SYSMENU | Создается окно с системным меню в области заголовка |
| WS\_THICKFRAME | Создается окно с рамкой, которая позволяет изменять его размеры |
| WS\_VISIBLE | Создается окно, которое сразу же является видимым |
| WS\_VSCROLL | Создается окно с вертикальной полосой прокрутки |

Параметры *nWidth* и *nHeight* определяют соответственно ширину и высоту окна в пикселях. Если параметр *nWidth* имеет значение CW\_USEDEFAULT, то значение *nHeight* игнорируется и операционная си стема сама определяет размер окна.

Девятый параметр, *hWndParent*, содержит дескриптор родительско го окна (окна-владельца). Следует заметить, что если между окнами

существует связь *родительское – дочернее*, дочернее окно всегда отображается только на поверхности родительского окна. Если окно не имеет родительского окна (обычно это главное окно приложения), па раметр *hWndParent* должен быть установлен в NULL.

Десятый параметр, *hMenu*, содержит дескриптор меню или иденти фикатор элемента управления, который содержится в сообщениях ро дительскому окну, поступающих от элемента управления. Интерпре тация значения этого параметра зависит от вида окна.

Параметр *hInstance* – дескриптор экземпляра приложения. Последний параметр, *lpParam*, может быть использован для пере дачи окну дополнительных данных в момент его создания. Если пара метр *lpParam* не используется, он должен принимать значение NULL. В случае успеха функция CreateWindowEx возвращает дескриптор созданного окна. Если по какой-то причине создать окно не удалось, то функция возвращает NULL.

В листинге 1.2 представлен пример создания главного окна и двух элементов управления, как показано на рис. 1.2.

*Листинг 1.2.* Пример создания окна с элементами управления

1 // *создание окна*

2 HWND **hWnd** = CreateWindowEx(0,

3 TEXT("MyWindowClass"), TEXT("Главное окно"),

4 WS\_OVERLAPPEDWINDOW,

5 CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0,

6 NULL, NULL, hInstance, NULL);

7

8 if (NULL != **hWnd**) // *если окно успешно создано*

9 {

10 // *создание элемента управления для созданного окна* 11 CreateWindowEx(WS\_EX\_CLIENTEDGE,

12 TEXT("Edit"), TEXT("Редактируемое поле"), 13 ES\_AUTOHSCROLL|ES\_LEFT|WS\_CHILD|WS\_VISIBLE, 14 10, 10, 200, 25,

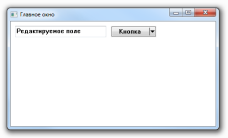
15 **hWnd**, (HMENU)**IDC\_EDIT1**, hInstance, NULL); 16

17 // *создание элемента управления для созданного окна* 18 CreateWindowEx(0,

19 TEXT("Button"), TEXT("Кнопка"),

20 BS\_CENTER|BS\_SPLITBUTTON|WS\_CHILD|WS\_VISIBLE, 21 220, 10, 100, 25,

22 **hWnd**, (HMENU)**IDC\_BUTTON1**, hInstance, NULL); 23 } // *if*

**

*Рис. 1.2.* Создание окна с элементами управления

Обратите внимание на то, что в этом примере для идентификации элементов управления используются две константы IDC\_EDIT1 и IDC\_BUTTON1, которые могут быть определены следующим образом:

1 #define IDC\_EDIT1 2001

2 #define IDC\_BUTTON1 2002

Для некоторых элементов управления общего пользования суще ствует альтернатива вызову функции CreateWindowEx – это вызов спе циальной функции создания элемента управления, которая в тоже вре мя может выполнять некоторую инициализацию. В табл. 1.7 приведе ны такие функции для некоторых элементов управления.

*Таблица 1.7.* Элементы управления и функции их создания

| **Элемент управления** | **Функция создания** |
| --- | --- |
| Панель инструментов (Toolbar) | CreateToolbarEx |
| Строка состояния (Status bar) | CreateStatusWindow |
| Счетчик или стрелки (Spin) | CreateUpDownControl |

**Диалоговые окна**

Различают *модальные* (modal) и *немодальные* (modeless) диалого вые окна. Как правило, модальные диалоговые окна используются ча ще, чем немодальные.

Модальным называется окно, которое не позволяет пользователю работать с другими окнами приложения до тех пор, пока работа с мо дальным окном не будет завершена. Однако пользователь может рабо тать с окнами других приложений. Немодальные диалоговые окна не требуют своего завершения для работы с другими окнами. Это значит,

что пользователь может свободно переключаться между диалоговым окном и другими окнами приложения.

***Создание диалогового окна***

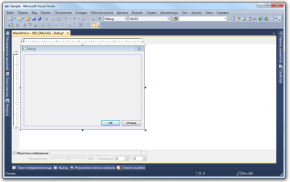
Диалоговое окно создается с использованием шаблона, который можно определить в файле ресурсов приложения, используя редактор диалоговых окон Visual Studio. Для определения шаблона диалогового окна нужно выполнить следующие действия:

1. В окне **Обозреватель решений** (**Solution Explorer**) выберите нужный проект.

2. В меню **Проект** (**Project**) выберите **Добавить ресурс** (**Add Re source**).

Откроется диалоговое окно **Добавление ресурса** (**Add Resource**). 3. В области **Тип ресурса** (**Resource type**) выберите **Dialog**. Нажмите кнопку **Создать** (**New**).

4. В открывшемся редакторе Visual Studio (рис. 1.3), отредактируйте и сохраните созданный шаблон диалогового окна.

*Рис. 1.3.* Шаблон диалогового окна в редакторе Visual Studio

В Win32 API имеются следующие функции для создания диалого вого окна:

INT\_PTR **DialogBox**(HINSTANCE *hInstance*, LPCTSTR *lpTemplate*, HWND *hWndParent*, DLGPROC *lpDialogFunc*);

INT\_PTR **DialogBoxParam**(HINSTANCE *hInstance*,

LPCTSTR *lpTemplate*, HWND *hWndParent*,

DLGPROC *lpDialogFunc*, LPARAM *dwInitParam*);

HWND **CreateDialog**(HINSTANCE *hInstance*, LPCTSTR *lpTemplate*, HWND *hWndParent*, DLGPROC lpDialogFunc);

HWND **CreateDialogParam**(HINSTANCE *hInstance*,

LPCTSTR *lpTemplate*, HWND *hWndParent*,

DLGPROC *lpDialogFunc*, LPARAM *dwInitParam*);

Первый параметр, *hInstance*, задает дескриптор модуля, который содержит шаблон диалогового окна.

Второй параметр, *lpTemplate*, определяет шаблон диалогового ок на. Чтобы задать этот параметр, можно использовать макрос MAKEINT RESOURCE, в котором следует указать идентификатор ресурса шаблона диалогового окна.

Третий параметр, *hWndParent*, – дескриптор окна, которое будет владеть диалоговым окном.

Четвертый параметр, *lpDialogFunc*, указывает на процедуру диало гового окна, которая обрабатывает отправляемые окну сообщения (по дробнее см. в разделе «Оконные сообщения»).

Параметр *dwInitParam* устанавливает значение, передаваемое в процедуру диалогового окна при инициализации диалогового окна. Первые две функции, DialogBox и DialogBoxParam, создают мо дальное диалоговое окно и возвращают управление только после за крытия этого окна. При успешном завершении эти функции возвра щают значение, указанное при завершении работы диалогового окна, в противном случае 0 или -1. Если параметр *hWndParent* определяет не допустимое значение, возвращаемое значение равно нулю. Функции CreateDialog и CreateDialogParam создают не модальное диалоговое окно и возвращают управление сразу же. В случае успеха возвращается дескриптор созданного диалогового окна, а в случае ошибки – NULL.

Диалоговые окна созданные, с помощью функции DialogBox или DialogBoxParam, должны уничтожаться функцией EndDialog, вызывае мой из процедуры диалогового окна.

BOOL **EndDialog**(HWND *hDlg*, INT\_PTR *nResult*);

Первый параметр, *hDlg*, – дескриптор диалогового окна. Второй параметр, *nResult*, определяет значение, возвращаемое функцией, ко торая создала диалоговое окно.

Если функция EndDialog завершается успешно, возвращается зна чение отличное от FALSE.

В следующем примере показано создание модального диалогового окна, с помощью функции DialogBox.

*Листинг 1.3.* Пример создания модального диалогового окна

1 INT\_PTR nDlgRes = DialogBox(hInstance,

2 MAKEINTRESOURCE(IDD\_DIALOG1), // *идентификатор шаблона* 3 hWnd, // *дескриптор окна владельца*

4 DialogProc); // *процедура диалогового окна*

5

6 if (IDOK == nDlgRes)

7 {

8 /\* *функция DialogBox вернула значение IDOK* \*/

9 } // *if*

***Окно сообщений***

Простейшим типом диалогового окна является окно сообщений. Окно сообщений содержит заголовок, текст сообщения, а также соче тание предопределенных пиктограмм и кнопок. Создавать окно сооб щений позволяет функция MessageBox:

int **MessageBox**(HWND *hWnd*, LPCTSTR *lpText*, LPCTSTR *lpCaption*, UINT *uType*);

Первый параметр, *hWnd*, задает дескриптор окна-владельца. Если этот параметр установлен в NULL, диалоговое окно не имеет владельца. Второй параметр, *lpText*, – указатель на строку, которая содержит текст выводимого сообщения. Можно использовать управляющий символ «\n», если необходимо сделать сообщение многострочным. Параметр *lpCaption* – указатель на строку, которая содержит заго ловок диалогового окна. Если этот параметр установлен в NULL, ис пользуется заданный по умолчанию заголовок «Ошибка» («Error»). Последний параметр, *uType*, определяет внешний вид и режим ра боты диалогового окна. Этот параметр может принимать одно или не сколько значений, перечисленных в табл. 1.8 – 1.11.

Для указания состава отображаемых кнопок используются значе ния, перечисленные в табл. 1.8.

*Таблица 1.8.* Флаги, определяющие состав кнопок

| **Флаг** | **Описание** |
| --- | --- |
| MB\_ABORTRETRYIGNORE | Диалоговое окно содержит три кнопки: **Прекратить** (**Abort**), **Повторить** (**Retry**) и **Пропустить** (**Ignore**) |
| MB\_CANCELTRYCONTINUE | Диалоговое окно содержит три кнопки: **Отменить** (**Cancel**), **Повторить** (**Retry**) и **Продолжить** (**Con tinue**) |

*Окончание табл. 1.8.*

| **Флаг** | **Описание** |
| --- | --- |
| MB\_HELP | К кнопкам диалогового окна добавляется кнопка **Справка** (**Help**). Когда пользователь щелкает по этой кнопке, окно-владелец получает сообщение WM\_HELP, а диалоговое окно продолжает работу |
| MB\_OK | Диалоговое окно содержит кнопку **OK**. Это значе ние используется по умолчанию |
| MB\_OKCANCEL | Диалоговое окно содержит две кнопки: **OK** и **Отме нить** (**Cancel**) |
| MB\_RETRYCANCEL | Диалоговое окно содержит две кнопки: **Повторить** (**Retry**) и **Отменить** (**Cancel**) |
| MB\_YESNO | Диалоговое окно содержит две кнопки: **Да** (**Yes**) и **Нет** (**No**) |
| MB\_YESNOCANCEL | Диалоговое окно содержит три кнопки: **Да** (**Yes**), **Нет** (**No**) и **Отменить** (**Cancel**) |

Чтобы указать отображаемую пиктограмму, используются значе ния, перечисленные в табл. 1.9.

*Таблица 1.9.* Флаги, определяющие отображаемую пиктограмму

| **Флаг** | **Пиктограмма** |
| --- | --- |
| MB\_ICONEXCLAMATION  MB\_ICONWARNING |  |
| MB\_ICONINFORMATION  MB\_ICONASTERISK |  |
| MB\_ICONASTERISK  MB\_ICONQUESTION |  |
| MB\_ICONSTOP  MB\_ICONERROR  MB\_ICONHAND |  |

Для указания кнопки по умолчанию, используются значения из табл. 1.10. *Кнопка по умолчанию* (default button) нажимается автомати чески при нажатии клавиши «Enter» или «Пробел».

*Таблица 1.10.* Флаги, определяющие кнопку по умолчанию

| **Флаг** | **Описание** |
| --- | --- |
| MB\_DEFBUTTON1 | Первая кнопка |
| MB\_DEFBUTTON2 | Вторая кнопка |

*Окончание табл. 1.10.*

| **Флаг** | **Описание** |
| --- | --- |
| MB\_DEFBUTTON3 | Третья кнопка |
| MB\_DEFBUTTON4 | Четвертая кнопка |

Чтобы указать правила поведения диалогового окна, используются значения, перечисленные в табл. 1.11. Есть и другие, реже применяе мые флаги, которые перечислены в документации Platform SDK.

*Таблица 1.11.* Флаги, определяющие поведение

| **Флаг** | **Описание** |
| --- | --- |
| MB\_APPLMODAL | Окно-владелец блокируется до тех пор, пока работа диалого вого окна не завершена. Это значение используется по умол чанию |
| MB\_SYSTEMMODAL | Сочетание MB\_APPLMODAL и MB\_TOPMOST |
| MB\_TASKMODAL | То же самое, что и MB\_APPLMODAL за исключением того, что блокируются все окна приложения, если диалоговое окно не имеет владельца |
| MB\_TOPMOST | Диалоговое окно всегда располагается поверх остальных окон, даже если оно неактивно |

Если функция MessageBox завершается ошибкой, возвращаемое значение равно нулю. В случае успеха функция возвращает одно из следующих значений:

− IDABORT – была нажата кнопка **Прекратить** (**Abort**); − IDCANCEL – была нажата кнопка **Отменить** (**Cancel**); − IDCONTINUE – была нажата кнопка **Продолжить** (**Continue**); − IDIGNORE – была нажата кнопка **Пропустить** (**Ignore**); − IDNO – была нажата кнопка **Нет** (**No**);

− IDOK – была нажата кнопка **OK**;

− IDRETRY – была нажата кнопка **Повторить** (**Retry**); − IDYES – была нажата кнопка **Да** (**Yes**).

В листинге 1.4 представлен пример создания изображенного на рис. 1.4 окна сообщений.

*Листинг 1.4.* Пример создания окна сообщений

1 int mbResult = MessageBox(hWnd, TEXT("Текст сообщения"), TEXT("Заголовок"), MB\_YESNOCANCEL | MB\_ICONINFORMATION | MB\_DEFBUTTON3);

2

3 switch (mbResult)

4 {

5 case IDYES: // *нажата кнопка Да*

6 break;

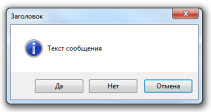
7 case IDNO: // *нажата кнопка Нет*

8 break;

9 case IDCANCEL: // *нажата кнопка Отмена*

10 break;

11 } // *switch*

**

*Рис. 1.4.* Пример окна сообщений

***Диалоговые окна общего пользования***

В Win32 API имеется набор функций, которые создают *диалоговые окна общего пользования* (common dialog box). В табл. 1.12 перечисле ны диалоговые окна общего пользования и функции их создания.

*Таблица 1.12.* Функции создания диалоговых окон общего пользования

| **Диалоговое окно** | **Функция создания** |
| --- | --- |
| Модальное диалоговое окно **Цвет** (**Color**) | ChooseColor |
| Модальное диалоговое окно **Шрифт** (**Font**) | ChooseFont |
| Немодальное диалоговое окно **Найти** (**Find**) | FindText |
| Модальное диалоговое окно **Открыть** (**Open**) | GetOpenFileName |
| Модальное диалоговое окно **Сохранить как** (**Save As**) | GetSaveFileName |
| Модальное диалоговое окно **Печать** (**Print**) | PrintDlg |
| Немодальное диалоговое окно **Заменить** (**Replace**) | ReplaceText |

Подробное описание диалоговых окон общего пользования можно найти в документации Platform SDK.

В следующем примере проиллюстрировано создание диалогового окна **Печать** (**Print**), изображенного на рис. 1.5:

*Листинг 1.5.* Пример создания диалогового окна для настройки печати

1 PRINTDLG pd = { sizeof(PRINTDLG) };

2

3 pd.Flags = PD\_ALLPAGES|PD\_USEDEVMODECOPIESANDCOLLATE; 4 pd.hwndOwner = hWnd; // *дескриптор окна-владельца* 5 pd.nCopies = 1; // *число копий*

6 pd.nFromPage = 15; // *начальная страница*

7 pd.nToPage = 48; // *конечная страница*

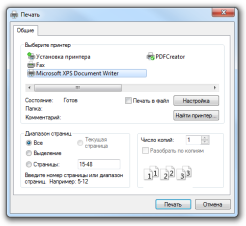
8 pd.nMinPage = 1; // *минимальное значение диапазона страниц* 9 pd.nMaxPage = 65535; // *максимальное значение диапазона страниц* 10

11 // *создание диалогового окна Печать*

12 if (PrintDlg(&pd) != FALSE)

13 {

14 /\* *Диалоговое окно закрыто*. *Нажата кнопка Печать* \*/ 15 } // *if*

*Рис. 1.5.* Диалоговое окно для настройки печати

**Функции работы с окнами**

Win32 API предоставляет широкий набор функций, которые рабо тают с созданными окнами. В табл. 1.13 перечислены наиболее упо требительные функции. Подробное описание этих и других функций, работающих с окнами, см. в документации Platform SDK.

*Таблица 1.13.* Функции для работы с окнами

| **Функция** | **Описание** |
| --- | --- |
| BringWindowToTop | Активизирует окно и переносит его поверх остальных окон, если оно находится позади |
| CheckDlgButton | Изменяет состояние элемента управления «переклю чатель» или «флажок» |
| CloseWindow | Сворачивает окно |
| DestroyWindow | Уничтожает окно и все его дочерние окна. Нельзя уничтожить окна других приложений |
| EnableWindow | Разрешает или запрещает окну ввод данных с клавиа туры или с помощью мыши |
| EnumChildWindows | Перечисляет дочерние окна |
| EnumWindows | Перечисляет все окна |
| FindWindow(Ex) | Возвращает дескриптор окна, найденного по имени окна и имени оконного класса |
| GetActiveWindow | Возвращает дескриптор активного окна |
| GetClassInfo | Возвращает информацию об оконном классе |
| GetClassLong | Возвращает значение указанного атрибута оконного класса |
| GetClassName | Возвращает имя класса окна |
| GetClientRect | Возвращает координаты клиентской области окна (в системе координат клиентской области) |
| GetDesktopWindow | Возвращает дескриптор окна рабочего стола |
| GetDlgCtrlID  GetWindowID | Возвращает идентификатор окна |
| GetDlgItem | Возвращает дескриптор элемента управления в окне |
| GetDlgItemInt | Возвращает текст, связанный с элементом управления в окне, преобразуя его в целочисленное значение |
| GetDlgItemText | Возвращает текст, связанный с элементом управления в окне |
| GetFirstChild | Возвращает дескриптор первого дочернего окна |
| GetFirstSibling | Возвращает дескриптор первого окна среди окон, принадлежащих к тому же окну |

*Продолжение табл. 1.13.*

| **Функция** | **Описание** |
| --- | --- |
| GetFocus | Возвращает дескриптор окна, имеющего фокус ввода |
| GetForegroundWindow | Возвращает дескриптор приоритетного окна (окна, с которым пользователь в настоящее время работает) |
| GetLastSibling | Возвращает дескриптор последнего окна среди окон, принадлежащих тому же окну |
| GetNextSibling | Возвращает дескриптор следующего соседнего окна, которое принадлежит тому же окну |
| GetParent | Возвращает дескриптор родительского окна или окна владельца |
| GetPrevSibling | Возвращает дескриптор предыдущего соседнего окна, которое принадлежит тому же окну |
| GetWindowExStyle | Возвращает расширенный стиль окна |
| GetWindowInfo | Возвращает информацию об окне |
| GetWindowInstance | Возвращает дескриптор экземпляра приложения, в котором создано окно |
| GetWindowLong | Возвращает значение указанного атрибута окна |
| GetWindowOwner | Возвращает дескриптор окна-владельца |
| GetWindowRect | Возвращает координаты окна (в экранной системе координат) |
| GetWindowStyle | Возвращает стиль окна |
| GetWindowText | Возвращает имя окна |
| GetWindowTextLength | Возвращает длину имени окна |
| IsChild | Проверяет, является ли окно дочерним окном по от ношению к указанному окну |
| IsDlgButtonChecked | Определяет состояние элемента управления «пере ключатель» или «флажок» |
| IsIconic  IsMinimized | Проверяет, свернуто ли окно |
| IsWindow | Определяет, идентифицирует ли дескриптор окна существующее окно |
| IsWindowEnabled | Проверяет, разрешен или запрещен ввод данных в окно |
| IsWindowVisible | Проверяет, состояние видимости окна |
| IsZoomed  IsMaximized | Проверяет, развернуто ли окно |
| SetActiveWindow | Активизирует окно |

*Окончание табл. 1.13.*

| **Функция** | **Описание** |
| --- | --- |
| SetClassLong | Изменяет атрибуты оконного класса |
| SetDlgItemInt | Изменяет текст элемента управления в окне на стро ковое представление указанного целочисленного зна чения |
| SetDlgItemText | Изменяет текст элемента управления в окне |
| SetFocus | Устанавливает в окне фокус клавиатуры |
| SetParent | Заменяет родительское окно |
| SetWindowLong | Изменяет атрибуты окна |
| SetWindowPos  MoveWindow | Изменяет размер и позицию окна |
| SetWindowText | Изменяет имя окна |
| ShowWindow | Устанавливает состояние отображения окна |

В листинге 1.6 представлен небольшой пример, в котором исполь зуются функции для работы с окнами – FindWindow и CloseWindow. *Листинг 1.6.* Пример использования функций для работы с окнами

1 // *поиск окна с заголовком "Калькулятор"*

2 HWND hWnd = FindWindow(NULL, TEXT("Калькулятор")); 3

4 if (NULL != hWnd) // *если окно успешно найдено*

5 {

6 CloseWindow(hWnd); // *свернуть окно*

7 } // *if*

***Отображение окна***

Для отображения на экране созданного окна используется функ ция ShowWindow:

BOOL **ShowWindow**(HWND *hWnd*, int *nCmdShow*);

Первый параметр, *hWnd*, задает дескриптор отображаемого окна, а второй параметр, *nCmdShow*, определяет, в каком виде будет показано окно. Для задания параметра *nCmdShow* можно использовать любое из значений, приведенных в табл. 1.14.

*Таблица 1.14.* Значения параметра *nCmdShow*

| **Значение** | **Описание** |
| --- | --- |
| SW\_HIDE | Скрыть указанное окно. Если окно активно, активизи руется другое окно |

*Окончание табл. 1.14.*

| **Значение** | **Описание** |
| --- | --- |
| SW\_MAXIMIZE | Развернуть указанное окно. Активное окно остается активным |
| SW\_MINIMIZE | Свернуть указанное окно. Если окно активно, активизи руется другое окно |
| SW\_RESTORE | Активизировать и отобразить указанное окно. Если окно свернуто или развернуто, оно восстанавливается в первоначальных размерах и позиции |
| SW\_SHOW | Активизировать и отобразить указанное окно. Окно отображается в последних размерах и позиции |
| SW\_SHOWDEFAULT | Выполняет заданное по умолчанию отображение ука занного окна |
| SW\_SHOWMAXIMIZED | Активизировать и развернуть указанное окно |
| SW\_SHOWMINIMIZED | Активизировать и свернуть указанное окно |
| SW\_SHOWMINNOACTIVE | Свернуть указанное окно. Активное окно остается ак тивным |
| SW\_SHOWNOACTIVATE | Отобразить указанное окно в последнем размере и по зиции. Активное окно остается активным |

Функция ShowWindow в случае успеха возвращает значение отлич ное от FALSE.

**Оконные сообщения**

Операционная система помещает каждое *оконное сообщение* (win dows message), адресованное окнам приложения, в *очередь сообщений* (message queue) этого приложения. Приложение извлекает сообщения из очереди в *цикле обработки сообщений* (message loop). После извле чения сообщения передаются *оконной процедуре* (window procedure), в которой выполняется обработка этих сообщений.

**Цикл обработки сообщений**

В каждом приложении Windows должен находиться цикл обработ ки сообщений, который обычно выглядит следующим образом:

*Листинг 1.7.* Цикл обработки сообщений

1 MSG msg;

2 BOOL bRet;

3

4 while ((bRet = GetMessage(&msg, NULL, 0, 0)) != FALSE) 5 {

6 if (bRet == -1)

7 {

8 /\* *обработка ошибки и возможно выход из цикла* \*/ 9 } // *if*

10 else

11 {

12 TranslateMessage(&msg);

13 DispatchMessage(&msg);

14 } // *else*

15 } // *while*

Структура MSG содержит информацию о сообщении, полученном из очереди сообщений. Эта структура определена в заголовочном фай ле WinUser.h следующим образом:

typedef struct tagMSG {

HWND *hwnd*; // *дескриптор окна*

UINT *message*; // *код (идентификатор) сообщения*

WPARAM *wParam*; // *дополнительная информация о сообщении* LPARAM *lParam*; // *дополнительная информация о сообщении* DWORD *time*; // *время создания сообщения*

POINT *pt*; // *позиция курсора (в экранных координатах)* // *в момент создания сообщения*

} MSG, \*PMSG, \*LPMSG;

Первое поле, *hwnd*, содержит дескриптор окна, которому адресова но полученное сообщение.

Второе поле, *message*, – код сообщения, который идентифицирует полученное сообщение. Для всех оконных сообщений в заголовочном файле WinUser.h определены символические константы, что облегча ет чтение и понимание программ.

Поля *wParam* и *lParam* содержат дополнительную информацию о полученном сообщении, которая зависит от кода сообщения. Последние два поля, *time* и *pt*, в дополнительных комментариях не нуждаются.

Извлечение сообщения из очереди осуществляется с помощью функции GetMessage:

BOOL **GetMessage**(LPMSG *lpMsg*, HWND *hWnd*, UINT *wMsgFilterMin*, UINT *wMsgFilterMax*);

Первый параметр, *lpMsg*, указывает на структуру MSG, в которую помещается информация об извлекаемом сообщении.

Второй параметр, *hWnd*, содержит дескриптор окна, для которого извлекается сообщение. Обычно этот параметр устанавливают в NULL, что позволяет извлекать сообщения для любого окна приложения.

Параметры *wMsgFilterMin* и *wMsgFilterMax* задают диапазон извле каемых сообщений. Если оба этих параметра принимают значение

равное нулю, то функция извлекает из очереди любое очередное со общение.

Функция GetMessage возвращает значение TRUE при получении лю бого сообщения, кроме WM\_QUIT. При получении сообщения WM\_QUIT, возвращается значение FALSE, что приводит к выходу из цикла обра ботки сообщений (см. листинг 1.7). В случае возникновения ошибки GetMessage возвращает значение –1.

Нужно отметить, что если в очереди нет сообщений, функция GetMessage, не возвращает управление программе, а ждет, когда сооб щение появится в очереди. Однако в некоторых приложениях может возникнуть необходимость выполнять фоновую обработку каких-либо данных, если в очереди нет сообщений. Для решения этой проблемы, Win32 API предоставляет функцию PeekMessage, которая проверяет наличие сообщения в очереди и при необходимости извлекает его.

Прототип функции PeekMessage практически идентичен прототипу функции GetMessage:

BOOL **PeekMessage**(LPMSG *lpMsg*, HWND *hWnd*, UINT *wMsgFilterMin*, UINT *wMsgFilterMax*, UINT *wRemoveMsg*);

Данная функция возвращает значение TRUE, если в очереди имеет ся сообщение, иначе – FALSE. Первые четыре параметра функции PeekMessage такие же, как и у функции GetMessage. Последний пара метр, *wRemoveMsg*, определяет, как именно сообщение извлекается из очереди. Для задания параметра *wRemoveMsg* можно использовать одно из следующих значений:

− PM\_NOREMOVE – сообщение остается в очереди;

− PM\_REMOVE – сообщение удаляется из очереди.

В листинге 1.8 представлен пример цикла обработки сообщений, в котором используется функция PeekMessage.

*Листинг 1.8.* Цикл обработки сообщений (без ожидания сообщения)

1 MSG msg;

2 BOOL bRet;

3

4 for (;;)

5 {

6 // *определяем наличие сообщений в очереди*

7 while (!PeekMessage(&msg, NULL, 0, 0, PM\_NOREMOVE)) 8 {

9 /\* *пока нет сообщений, выполняется какая-нибудь работа* \*/ 10 } // *while*

11

12 // *извлекаем сообщение из очереди*

13 bRet = GetMessage(&msg, NULL, 0, 0);

14

15 if (bRet == -1)

16 {

17 /\* *обработка ошибки и возможно выход из цикла* \*/ 18 } // *if*

19 else if (FALSE == bRet)

20 {

21 break; // *получено WM\_QUIT, выход из цикла* 22 } // *if*

23 else

24 {

25 TranslateMessage(&msg);

26 DispatchMessage(&msg);

27 } // *else*

28 } // *for*

В теле цикла обработки сообщений можно увидеть вызовы функ ций TranslateMessage и DispatchMessage, которые имеют следующие прототипы:

BOOL **TranslateMessage**(const MSG \**lpMsg*);

LRESULT **DispatchMessage**(const MSG \**lpmsg*);

Функция TranslateMessage используется только в тех случаях, ко гда необходимо обрабатывать ввод с клавиатуры. Дело в том, что в Windows реализована двух уровневая схема обработки сообщений от клавиатуры, в которой сначала из очереди извлекается сообщение, содержащее *виртуальный код* (virtual key) нажатой клавиши, а затем сообщение, содержащее код символа (ANSI или Unicode) этой клави ши. Сообщения с виртуальными кодами клавиш генерируются опера ционной системой, тогда как сообщения с кодами символов помеща ются в очередь функцией TranslateMessage. Например, если будет по лучено сообщение WM\_KEYDOWN (была нажата клавиша), в котором со держится виртуальный код нажатой клавиши, то функция TranslateMessage добавит в начало очереди сообщение WM\_CHAR, кото рое содержит код символа нажатой клавиши.

В случае если сообщение (с кодом символа нажатой клавиши) бы ло добавлено в очередь функция TranslateMessage возвращает TRUE, иначе – FALSE. Следует также отметить, что функция TranslateMessage не изменяет сообщение, указанное параметром *lpMsg*.

Функция DispatchMessage передает полученное сообщение окон ной процедуре того окна, которому оно было адресовано. Это может

быть оконная процедура окна (созданного приложением) или оконная процедура одного из элементов управления общего пользования. Функция DispatchMessage возвращает значение, которое возвра щает оконная процедура и которое, как правило, игнорируется.

***Цикл обработки сообщений диалоговых окон***

Чтобы обрабатывать сообщения модального диалогового окна, операционная система запускает собственный цикл обработки сооб щений, временно забирая у приложения управление очередью сооб щений.

Обработка сообщений немодального диалогового окна, напротив, выполняется в цикле обработки сообщений приложения, с помощью вызова функции IsDialogMessage, например, следующим образом:

*Листинг 1.9.* Обработка сообщений немодального диалогового окна

1 MSG msg;

2 BOOL bRet;

3

4 while ((bRet = GetMessage(&msg, NULL, 0, 0)) != FALSE) 5 {

6 if (bRet == -1)

7 {

8 /\* *обработка ошибки и возможно выход из цикла* \*/ 9 } // *if*

10 else if (**IsDialogMessage**(hDlg, &msg) == FALSE) 11 {

12 TranslateMessage(&msg);

13 DispatchMessage(&msg);

14 } // *if*

15 } // *while*

В этом примере используется глобальная переменная hDlg, кото рая содержит дескриптор немодального диалогового окна. Функция IsDialogMessage определяет, предназначено ли сообще ние указанному диалоговому окну и, если это так, обрабатывает его.

BOOL **IsDialogMessage**(HWND *hDlg*, LPMSG *lpMsg*);

Первый параметр, *hDlg*, дескриптор диалогового окна, сообщение которого нужно обработать. Второй параметр, *lpMsg*, указывает на со общение, которое нужно обработать.

Если сообщение было обработано функцией IsDialogMessage, воз вращается значение TRUE, иначе – FALSE.

**Оконная процедура**

Обработкой оконных сообщений в приложении занимается специ альная функция, называемая *оконной процедурой*. Для каждого окна создается своя копия оконной процедуры, что позволяет использовать в ней статические переменные.

Имя оконной процедуры может быть любым. Оконная процедура имеет следующую сигнатуру:

LRESULT CALLBACK **WindowProc**(HWND *hWnd*, UINT *uMsg*, WPARAM *wParam*, LPARAM *lParam*);

Первый параметр, *hWnd*, содержит дескриптор окна, получающего сообщение. Второй параметр, *uMsg*, – код сообщения. Параметры *wParam* и *lParam* содержат дополнительную информацию, которая зави сит от кода полученного сообщения.

В теле оконной процедуры должна выполняться обработка полу ченных сообщений. Все сообщения, которые не обрабатываются окон ной процедурой, должны передаваться в функцию DefWindowProc, ко торая выполняет стандартную обработку сообщений. В этом случае

оконная процедура должна вернуть то значение, которое вернет функ ция DefWindowProc.

Функция DefWindowProc фактически является оконной процедурой по умолчанию. Поэтому прототип данной функции имеет туже сигна туру, что и оконная процедура:

LRESULT CALLBACK **DefWindowProc**(HWND *hWnd*, UINT *uMsg*, WPARAM *wParam*, LPARAM *lParam*);

В листинге 1.10 представлен пример оконной процедуры, которая обрабатывает сообщения WM\_CREATE и WM\_DESTROY, а остальные передает в функцию DefWindowProc.

*Листинг 1.10.* Пример оконной процедуры

1 LRESULT CALLBACK WindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

2 {

3 switch (uMsg)

4 {

5 case **WM\_CREATE**: // *создание окна*

6 /\* *здесь выполняется обработка сообщения* \*/ 7 return 0;

8 case **WM\_DESTROY**: // *уничтожение окна*

9 /\* *здесь выполняется обработка сообщения* \*/ 10 return 0;

11 } // *switch*

12

13 // *передача необработанного сообщения*

14 // *оконной процедуре по умолчанию*

15 return **DefWindowProc**(hWnd, uMsg, wParam, lParam); 16 } // *WindowProc*

***Процедура диалогового окна***

Каждое диалоговое окно принадлежит предопределенному окон ному классу (см. табл. 1.3), которому соответствует своя оконная про цедура. Приложения не имеют прямого доступа к этой оконной проце дуре, но могут использовать так называемую *процедуру диалогового*

*окна* (dialog procedure). Эта процедура похожа на обычную оконную процедуру, но в отличие от оконной процедуры, процедура диалогово го окна никогда не вызывает функцию DefWindowProc. Вместо этого, она возвращает значение TRUE, если она обрабатывает оконное сооб щение, или FALSE, если она этого не делает.

Имя процедуры диалогового окна, как имя оконной процедуры, может быть любым. Процедура диалогового окна имеет следующую сигнатуру:

INT\_PTR CALLBACK **DialogProc**(HWND *hwndDlg*, UINT *uMsg*, WPARAM *wParam*, LPARAM *lParam*);

Параметры процедуры диалогового окна такие же, как у оконной процедуры. Первый параметр, *hwndDlg*, содержит дескриптор диалого вого окна, получающего сообщение. Второй параметр, *uMsg*, – код со общения. Параметры *wParam* и *lParam* содержат дополнительную ин формацию, которая зависит от кода полученного сообщения.

Если процедура диалогового окна обрабатывает сообщение, кото рое требует специального возвращаемого значения, то она может пе редать это значение через макрос SetDlgMsgResult, определенный в заголовочном файле WindowsX.h:

SetDlgMsgResult(*hwnd*, *msg*, *result*)

Первый параметр, *hwnd*, определяет дескриптор диалогового окна. Второй параметр, *msg*, идентифицирует сообщение. Третий параметр, *result*, задает значение, которая должна вернуть процедура диалого вого окна.

В этом случае процедура диалогового окна должна вернуть то зна чение, которое вернет макрос SetDlgMsgResult. Подробнее о том, ка ким образом процедура диалогового окна возвращает значения см. в документации Platform SDK.

В листинге 1.11 представлен пример процедуры диалогового окна, которая обрабатывает сообщения WM\_INITDIALOG и WM\_DESTROY. *Листинг 1.11.* Пример процедуры диалогового окна

1 INT\_PTR CALLBACK DialogProc(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)

2 {

3 switch (uMsg)

4 {

5 case **WM\_INITDIALOG**: // *инициализация диалогового окна* 6 /\* *здесь выполняется обработка сообщения* \*/ 7 return SetDlgMsgResult(hwndDlg, WM\_INITDIALOG, TRUE); 8 case **WM\_DESTROY**: // *уничтожение диалогового окна* 9 /\* *здесь выполняется обработка сообщения* \*/ 10 return TRUE;

11 } // *switch*

12

13 return FALSE;

14 } // *DialogProc*

**Часто используемые сообщения**

В этом разделе рассмотрены оконные сообщения, обработка кото рых в оконной процедуре встречается наиболее часто. Полный пере чень оконных сообщений, а также их описание, можно найти в доку ментации Platform SDK.

***WM\_CLOSE***

Сообщение WM\_CLOSE уведомляет окно о том, что оно должно быть закрыто. Приложение, обрабатывая это сообщение, может запросить у пользователя подтверждение, перед разрушением окна, и при под тверждении выбора вызвать функцию DestroyWindow.

Параметры *wParam* и *lParam* не используются. Если это сообщение обрабатывается, оконная процедура должна вернуть значение 0. Сообщение WM\_CLOSE может обрабатываться, например, следую щим образом:

*Листинг 1.12.* Пример обработки сообщения WM\_CLOSE

1 case WM\_CLOSE:

2 {

3 int mbResult = MessageBox(hWnd, TEXT("Хотите завершить работу?"), TEXT("Выход"), MB\_YESNO | MB\_ICONQUESTION | MB\_DEFBUTTON2);

4

5 if (IDYES == mbResult) // *пользователь выбрал «Да»*

6 {

7 DestroyWindow(hWnd); // *уничтожаем окно* 8 } // *if*

9 }

10 return 0;

По умолчанию функция DefWindowProc вызывает DestroyWindow, чтобы разрушить окно.

***WM\_COMMAND***

Сообщение WM\_COMMAND отправляется окну, когда пользователь вы бирает пункт в меню окна, когда элемент управления общего пользо вания отправляет уведомление своему родительскому окну, или когда транслируется нажатие быстрых клавиш.

Старшее слово параметра *wParam* определяет код уведомления, ес ли сообщение передано элементом управления. Если сообщение, пере дано в результате трансляции быстрых клавиш, это значение равно 1. Если же сообщение от меню, то значение равно 0. Младшее слово па раметра *wParam* определяет идентификатор пункта меню, элемента

управления или быстрой клавиши.

Получить значения, запакованные в параметре *wParam*, можно с помощью двух макросов LOWORD и HIWORD, которые возвращают соот ветственно младшее слово и старшее слово.

Параметр *lParam* – дескриптор элемента управления, посылающего сообщение. Может принимать значение NULL.

Если это сообщение обрабатывается, оконная процедура должна вернуть значение 0.

В следующем примере продемонстрирована обработка уведомле ния BN\_CLICKED (была нажата кнопка) для двух элементов управления «кнопка» (button).

*Листинг 1.13.* Пример обработки сообщения WM\_COMMAND

1 case WM\_COMMAND:

2 {

3 // *идентификатор элемента управления*

4 WORD id = LOWORD(wParam);

5 // *код уведомления*

6 WORD codeNotify = HIWORD(wParam);

7

8 if (BN\_CLICKED == codeNotify)

9 {

10 switch (id)

11 {

12 case IDC\_BUTTON1: // *обработка нажатия кнопки 1* 13 return 0;

14 case IDC\_BUTTON2: // *обработка нажатия кнопки 2* 15 return 0;

16 } // *switch*

17 } // *if*

18 }

19 break;

***WM\_CREATE***

Сообщение WM\_CREATE отправляется окну, создаваемому функцией CreateWindowEx или CreateWindow. (Отправление происходит, перед тем как функция, создавшая окно, возвращает значение.) Оконная проце дура окна принимает это сообщение после создания окна, но перед отображением.

Параметр *wParam* не используется. Параметр *lParam* указывает на структуру CREATESTRUCT, которая содержит информацию о создаваемом окне. Описание структуры CREATESTRUCT см. в документации Platform SDK.

Если это сообщение обрабатывается, оконная процедура должна вернуть значение 0. Если оконная процедура вернет –1, то окно будет уничтожено, а функция CreateWindow(Ex) вернет NULL.

***WM\_DESTROY***

Сообщение WM\_DESTROY отправляется уничтожаемому окну, после того, как оно исчезает с экрана. Параметры *wParam* и *lParam* не исполь зуются.

Если это сообщение обрабатывается, оконная процедура должна вернуть значение 0.

***WM\_INITDIALOG***

Сообщение WM\_INITDIALOG отправляется диалоговому окну непо средственно перед тем, как оно будет отображено на экране. Обработ ка этого сообщения позволяет выполнить инициализацию данных, свя занных с диалоговым окном.

Параметр *wParam* представляет собой дескриптор элемента управ ления, который по умолчанию принимает фокус клавиатуры, если процедура диалогового окна вернет значение TRUE.

Параметр *lParam* определяет дополнительные данные инициализа ции. Эти данные передаются, как параметр *lParam* при вызове функций DialogBoxParam или CreateDialogParam. Этот параметр принимает зна чение NULL, если для создания использовались другие функции.

Если процедура диалогового окна возвращает значение TRUE, эле мент управления, дескриптор которого указан в параметре *wParam*, ав томатически принимает фокус клавиатуры. Если в процедуре диалого вого окна для какого-либо элемента управления устанавливается фо кус клавиатуры, используя функцию SetFocus, то должно возвращать ся значение FALSE.

***WM\_MOVE***

Сообщение WM\_MOVE отправляется окну после того, как оно будет перемещено. Параметр *wParam* не используется, а параметр *lParam* со держит координаты левого и верхнего угла окна. Младшее слово па раметра *lParam* содержит *x*-координату, а старшее слово содержит *y* координату. Координаты указываются в экранных координатах для перекрывающих и всплывающих окон, а для дочерних окон в коорди натах клиентской области родительского окна.

Получить *x*- и *y*-координату можно с помощью двух макросов, определенных в заголовочном файле WindowsX.h – соответственно GET\_X\_LPARAM и GET\_Y\_LPARAM. Следующий пример демонстрирует, как это можно сделать:

1 WORD x = GET\_X\_LPARAM(lParam);

2 WORD y = GET\_Y\_LPARAM(lParam);

Если сообщение WM\_MOVE обрабатывается, оконная процедура должна вернуть значение 0.

***WM\_NOTIFY***

Сообщение WM\_NOTIFY отправляется окну элементом управления общего пользования, когда произошло некоторое событие или элемент управления требует некоторой информации.

Параметр *wParam* идентифицирует элемент управления. Однако нет гарантии, что этот идентификатор будет уникальным. Поэтому чтобы идентифицировать элемент управления следует использовать одно из полей структуры NMHDR (*hwndFrom* или *idFrom*), на которую указывает параметр *lParam*. Структура NMHDR описывается следующим образом:

typedef struct tagNMHDR {

HWND *hwndFrom*; // *дескриптор окна элемента управления* UINT\_PTR *idFrom*; // *идентификатор элемента управления* UINT *code*; // *код уведомления*

} NMHDR;

Первое поле, *hwndFrom*, содержит дескриптор окна элемента управ ления, от которого пришло уведомление. Второе поле, *idFrom*, – это

идентификатор этого элемента управления. Третье поле, *code*, содер жит код уведомления. Для каждого элемента управления существуют свои специфические уведомления (см. в документации Platform SDK).

Для некоторых уведомлений параметр *lParam* указывает на другую структуру, имеющую NMHDR в качестве первого поля.

В следующем примере продемонстрирована обработка уведомле ния NM\_CLICK (щелчок левой кнопкой мыши) для элемента управления «список просмотра» (list view):

*Листинг 1.14.* Пример обработки сообщения WM\_NOTIFY

1 case WM\_NOTIFY:

2 {

3 LPNMHDR lpnmhdr = (LPNMHDR)lParam;

4

5 switch (lpnmhdr->code)

6 {

7 case NM\_CLICK:

8 if (lpnmhdr->idFrom == IDC\_LISTVIEW1) 9 {

10 LPNMITEMACTIVATE lpnmitem =

(LPNMITEMACTIVATE)lpnmhdr;

11 /\* *обработка уведомления* \*/

12 return 0;

13 } // *if*

14 } // *switch*

15 }

16 break;

***WM\_SIZE***

Сообщение WM\_SIZE отправляется окну после изменения его разме ра. Параметр *wParam* может принимать одно из следующих значений:

− SIZE\_MAXHIDE – сообщение было послано всем всплывающим ок нам после того, как другое окно было развернуто;

− SIZE\_MAXIMIZED – окно было развернуто;

− SIZE\_MAXSHOW – сообщение было послано всем всплывающим ок нам после того, как другое окно было восстановлено до прежних размеров;

− SIZE\_MINIMIZED – окно было свернуто;

− SIZE\_RESTORED – размеры окна были изменены, но окно не было свернуто или развернуто.

Младшее слово параметра *lParam* содержит новую ширину кли ентской области. Старшее слово параметра *lParam* содержит новую

высоту клиентской области. Получить ширину и высоту из параметра *lParam* можно с помощью макросов LOWORD и HIWORD.

Если это сообщение обрабатывается, оконная процедура должна вернуть значение 0.

**Отправка сообщений**

В Win32 API имеется набор функций, предназначенных для от правки оконных сообщений. Основными из них являются PostMessage и SendMessage:

BOOL **PostMessage**(HWND *hWnd*, UINT *Msg*, WPARAM *wParam*, LPARAM *lParam*);

LRESULT **SendMessage**(HWND *hWnd*, UINT *Msg*, WPARAM *wParam*, LPARAM *lParam*);

Первый параметр, *hWnd*, содержит дескриптор окна, оконная про цедура которого примет сообщение. Если этот параметр принимает значение равное HWND\_BROADCAST, то сообщение отправляется всем ок нам верхнего уровня (такой тип сообщений не отправляется дочерним окнам).

Второй параметр, *uMsg*, – код сообщения, которое будет отправле но. Для отправки собственных сообщений в пределах оконного класса, можно использовать любое значение в диапазоне от WM\_USER до 0x7FFF.

Последние параметры, *wParam* и *lParam*, содержат дополнительную информацию, которая зависит от оконного сообщения. Основное различие между этими двумя функциями состоит в том, что функция PostMessage помещает сообщение в очередь сообщений и возвращает управление без ожидания обработки этого сообщения, то гда как функция SendMessage вызывает оконную процедуру заданного окна и не возвращает управление до тех пор, пока оконная процедура не обработает переданное сообщение. В первом случает речь идет о *синхронных* сообщениях, а во втором – об *асинхронных* сообщениях. Если функция PostMessage завершает успешно, возвращается зна чение отличное от FALSE.

Функция SendMessage возвращает значение, которое вернула окон ная процедура в результате обработки сообщения.

***Отправка сообщений элементам управления***

После создания элемента управления общего пользования прило жение управляет его действиями, отправляя необходимые сообщения при помощи функции SendMessage. Кроме общих оконных сообщений таких, как, например, WM\_SETFONT и WM\_SETTEXT, каждый элемент управ-

ления поддерживает свои специфические сообщения (см. в документа ции Platform SDK).

Например, для задания максимальной длины вводимого текста в элементе управления «редактируемом поле» (edit) с дескриптором ок на hwndEdit необходимо отправить этому элементу управления сооб щение EM\_LIMITTEXT:

SendMessage(hwndEdit, EM\_LIMITTEXT, 255, 0);

Альтернативой вызовам функции SendMessage является использо вание макросов, определенных в заголовочных файлах CommCtrl.h и WindowsX.h. Например, чтобы отправить элементу управления со общение EM\_LIMITTEXT можно использовать макрос Edit\_LimitText:

Edit\_LimitText(hwndEdit, 255);

Не трудно заметить, что программный код с макросом значитель но проще для чтения и понимания.

К сожалению, как видно из табл. 1.15, в Win32 API имеются мак росы не для всех элементов управления общего пользования.

*Таблица 1.15.* Префиксы макросов для элементов управления

| **Элемент управления** | **Оконный класс** | **Префикс**  **макроса** |
| --- | --- | --- |
| Анимационное изображение (Animate) | SysAnimate32 | Animate\_ |
| Выпадающий список (Combo box) | ComboBox  ComboBoxEx32 | ComboBox\_ |
| Дерево просмотра (Tree view) | SysTreeView32 | TreeView\_ |
| Заголовок списка просмотра (Header) | SysHeader32 | Header\_ |
| Закладки (Tab control) | SysTabControl32 | TabCtrl\_ |
| Календарь (Month calendar) | SysMonthCal32 | MonthCal\_ |
| Кнопка (Button), Переключатель  (Radio button), Флажок (Check box) | Button | Button\_ |
| Надпись (Static text) | Static | Static\_ |
| Поле ввода даты или времени  (Data time picker) | SysDateTimePick32 | DateTime\_ |
| Полоса прокрутки (Scroll bar) | ScrollBar | ScrollBar\_ |
| Редактируемое поле (Edit box) | Edit | Edit\_ |
| Список (List box) | ListBox | ListBox\_ |
| Список просмотра (List view) | SysListView32 | ListView\_ |

Кроме того, для отправки сообщений элементам управления мож но использовать функцию SendDlgItemMessage:

LRESULT **SendDlgItemMessage**(HWND *hDlg*, int *nIDDlgItem*, UINT *Msg*, WPARAM *wParam*, LPARAM *lParam*);

Первый параметр, *hDlg*, содержит дескриптор окна, которому при надлежит элемент управления. Второй параметр, *nIDDlgItem*, задает идентификатор элемента управления. Последние три параметра (*Msg*, *wParam* и *lParam*) аналогичны параметрам функции SendMessage.

Функция SendDlgItemMessage работает точно так же, как и функ ция SendMessage, то есть не возвращает управление до тех пор, пока оконная процедура не обработает сообщение.

***Отправка WM\_QUIT***

Для отправки сообщения WM\_QUIT в Win32 API используется функ ция PostQuitMessage:

void **PostQuitMessage**(int *nExitCode*);

Параметр *nExitCode* определяет код завершения. Значение этого параметра будет записано в поле *wParam* структуры MSG. Не следует посылать сообщение WM\_QUIT с помощью функций PostMessage или SendMessage.

Нужно отметить, что сообщение WM\_QUIT не может быть получено оконной процедурой, т.к. оно не связано с каким-либо окном. Только функции GetMessage и PeekMessage могут его получить.

**Распаковщики сообщений**

В заголовочном файле WindowsX.h определены макросы *распа ковщики сообщений* (message crackers), которые упрощают обработку сообщений в оконной процедуре. При их использовании придется по стоянно заглядывать в исходный текст файла WindowsX.h, так как в документации Platform SDK распаковщики не описаны.

Распаковщики сообщений позволяют обойти ряд трудностей, ко торые могут возникнуть при разработке оконной процедуры. Во первых, как правило, программный код оконной процедуры (кроме простейших примеров) разрастается до огромных размеров, так что становится крайне трудно его читать и отлаживать. Во-вторых, в окон ной процедуре может сосредотачиваться большое количество пере менных, часть из которых используется для обработки конкретного сообщения, тогда как остальные в этот момент не используются, бес полезно занимая место в памяти.

В WindowsX.h для большинства сообщения вида WM\_XXX опреде лен свой макрос с именем HANDLE\_WM\_XXX. Например, для сообщения WM\_CREATE определен следующий макрос:

HANDLE\_WM\_CREATE(*hwnd*, *wParam*, *lParam*, *fn*)

Параметры всех таких макросов одинаковые. Параметр *hwnd* – это дескриптор окна, получающего сообщение. Параметры *wParam* и *lParam* содержат дополнительную информацию. Последний параметр, *fn*, – имя функции, которая будет обрабатывать сообщение.

Макрос распаковщика сообщения должен использоваться внутри конструкции switch оконной процедуры. Например, следующий фраг мент программного кода:

1 case WM\_CREATE:

2 return HANDLE\_WM\_CREATE(hWnd, wParam, lParam, OnCreate);

в результате раскрытия представляет собой следующий программный код (подробнее см. исходный текст файла WindowsX.h):

1 case WM\_CREATE:

2 return ((**OnCreate**)((**hWnd**),(LPCREATESTRUCT)(**lParam**)) ? 0L : (LRESULT)-1L);

Таким образом, можно видеть, что макрос распаковщика сообще ния осуществляет распаковку параметров, вызов функции-обработчика сообщения и анализ возвращаемого результата.

Чтобы распаковщик правильно распознал функцию-обработчика, она должна иметь определенную сигнатуру, которая зависит от кода сообщения. В WindowsX.h перед определением соответствующего макроса приводится сигнатура функции-обработчика:

1 /\* *BOOL Cls\_OnCreate(HWND hwnd, LPCREATESTRUCT lpCreateStruct)* \*/ 2 #define **HANDLE\_WM\_CREATE**(hwnd, wParam, lParam, fn) \ 3 ((fn)((hwnd), (LPCREATESTRUCT)(lParam)) ? 0L : (LRESULT)-1L)

Из этого примера видно, что функция-обработчик сообщения WM\_CREATE имеет следующую сигнатуру:

BOOL **Cls\_OnCreate**(HWND *hwnd*, LPCREATESTRUCT *lpCreateStruct*);

Имя функции-обработчика не обязательно должно быть таким, как указано в файле WindowsX.h, оно может быть любым. Кроме того, в файле WindowsX.h определен небольшой макрос HANDLE\_MSG, который упрощает использование распаковщиков сообще ний в конструкции switch оконной процедуры:

HANDLE\_MSG(*hwnd*, *message*, *fn*)

Здесь параметр *hwnd* – дескриптор окна, получающего сообщение. Параметр *message* – код сообщения. Последний параметр, *fn*, – имя функции-обработчика.

Этот макрос используется в конструкции switch оконной процеду ры следующим образом:

1 switch (uMsg)

2 {

3 HANDLE\_MSG(hWnd, WM\_CREATE, OnCreate);

4 HANDLE\_MSG(hWnd, WM\_DESTROY, OnDestroy);

5 } // *switch*

***Упаковка сообщений***

В заголовочном файле WindowsX.h также определены макросы FORWARD\_WM\_XXX, которые решают обратную задачу – принимают дан ные сообщения, упаковывают их в параметры *wParam* и *lParam*, а затем вызывают функцию, указанную в качестве аргумента макроса, переда вая ей упакованные параметры. Например, для сообщения WM\_CREATE определен следующий макрос:

FORWARD\_WM\_CREATE(*hwnd*, *lpCreateStruct*, *fn*)

В качестве вызываемой функции может использоваться функция PostMessage, SendMessage или любая другая функция, имеющая анало гичный набор параметров.

**Приложения Win32**

Приложение Win32 (Win32 application) основано на *графическом интерфейсе пользователя* (Graphical User Interface, GUI). Такие при ложения создают окна, имеют меню, взаимодействуют с пользовате лем через диалоговые окна и т.п. Почти все стандартные приложения Windows (такие как «Блокнот», «Калькулятор» и др.) являются прило жениями Win32.

Вместе с тем различие между приложением Win32 и консольным приложением весьма условно. Можно, например, создать консольное приложение, в котором будут отображаться различные окна. В то же время можно создать приложение Win32, способное выводить данные в стандартный поток вывода, как это делает консольное приложение.

**Функция WinMain**

По аналогии с консольными приложениями C/C++, каждое при ложение Win32 должно иметь главную функцию – WinMain. При напи сании приложения, которое использует набор символов Unicode, мож но использовать wWinMain.

Можно также использовать макрос \_tWinMain, который объявлен в заголовочном файле tchar.h. В зависимости от определения константы \_UNICODE используется либо WinMain, либо wWinMain.

int WINAPI **WinMain**(HINSTANCE *hInstance*,

HINSTANCE *hPrevInstance*, LPSTR *lpCmdLine*, int *nCmdShow*);

int WINAPI **wWinMain**(HINSTANCE *hInstance*,

HINSTANCE *hPrevInstance*, LPWSTR *lpCmdLine*, int *nCmdShow*);

int WINAPI **\_tWinMain**(HINSTANCE *hInstance*,

HINSTANCE *hPrevInstance*, LPTSTR *lpCmdLine*, int *nCmdShow*);

Параметр *hInstance* – дескриптор экземпляра приложения, кото рый система присваивает запущенному приложению. Параметр *hPrev Instance* остался от Windows 3.1 для совместимости версий, в Win32 этот параметр не используется и поэтому всегда установлен в NULL. Параметр *lpCmdLine* указывает на строку, которая содержит аргументы приложения, при запуске в режиме командной строки. Параметр *nCmdShow* содержит значение (см. табл. 1.14), которое может быть пере дано функции ShowWindow.

***Точка входа приложения Win32***

Если в качестве главной функции используется WinMain, точкой входа при запуске приложения Windows является специальная функ ция WinMainCRTStartup. Функция WinMainCRTStartup инициализирует различные библиотеки C/C++, загружает необходимые DLL и создает и все глобальные переменные. Когда все это будет сделано, она вызы вает функцию WinMain.

Если же в качестве главной функции используется wWinMain, то точкой входа является функция wWinMainCRTStartup, которая вызывает wWinMain, но в остальном она делает все тоже, что и WinMainCRTStartup.

После того как функция WinMain (wWinMain) вернет управление, функция WinMainCRTStartup (wWinMainCRTStartup) удалит все глобаль ные переменные, выгрузит DLL и освободит ресурсы, выделенные для различных библиотек C/C++.

**Пользовательский интерфейс**

Каждая новая версия Windows предлагает улучшенный вариант пользовательского интерфейса. Помимо того, что в библиотеку ComCtl32.dll добавляются новые элементы управления общего поль зования, изменяется внешний вид уже существующих элементов управления. Например, на рис. 1.6 проиллюстрировано как изменялся внешний вид элемента управления «кнопка».

***а б в***

*Рис. 1.6.* Кнопка: *а* – традиционное оформление;

*б* – оформление в Windows XP; *в* – оформление в Windows Vista

Операционная система Windows XP и более новые версии исполь зуют библиотеку ComCtl32.dll версии 6. Все более ранние версии Windows используют ComCtl32.dll версии 5 с традиционным стилем оформления элементов управления.

По умолчанию приложения Win32, создаваемые в Visual C++, ис пользуют традиционный стиль оформления элементов управления. Чтобы использовать улучшенный стиль оформления элементов управ ления, в проект приложения нужно включить специальный ресурс – манифест. *Манифест приложения* (application manifest) представляет собой документ в формате XML, содержащий всю информацию, необ ходимую для взаимодействия приложения и библиотеки ComCtl32.dll версии 6. Пример файла манифеста представлен в листинге 1.15.

*Листинг 1.15.* Пример файла манифеста

1 <?xml version='1.0' encoding='UTF-8' standalone='yes'?> 2 <assembly xmlns='urn:schemas-microsoft-com:asm.v1' manifestVersion='1.0'>

3 <assemblyIdentity name='CompanyName.ProductName.YourApp' processorArchitecture='x86' version='1.0.0.0' type='win32' /> 4 <description>Your application description here.</description> 5 <dependency>

6 <dependentAssembly>

7 <assemblyIdentity type='win32'

name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='x86' publicKeyToken='6595b64144ccf1df' language='\*' />

8 </dependentAssembly>

9 </dependency>

10 </assembly>

Чтобы включить файл манифеста в проект Visual C++ выполните следующие действия:

1. Откройте решение, содержащее нужный проект (если это не было сделано заранее).

2. В меню **Проект** (**Project**) выберите **Свойства** (**Settings**). Откроется диалоговое окно **Страницы свойств** (**Property Pages**).

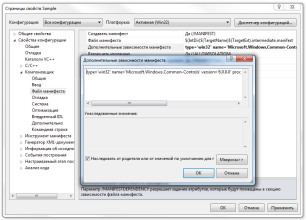
3. В открывшемся окне выберите **Свойства конфигурации** (**Config uration Properties**) → **Компоновщик** (**Linker**) → **Файл манифе ста** (**Manifest File**).

4. Отредактируйте значение параметра **Создавать манифест** (**Gen erate Manifest**), выбрав в нем **/MANIFEST**.

5. Отредактируйте значение параметра **Дополнительные зависимо сти манифеста** (**Additional Manifest Dependencies**), как показано на рис. 1.7, указав следующее значение (в одну строку):

type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='x86'

publicKeyToken='6595b64144ccf1df' language='\*'

6. Нажмите кнопку **OK**, чтобы сохранить изменения. *Рис. 1.7.* Добавление манифеста в проект Visual C++

**Ресурсы приложения Win32**

Ресурсы являются составной часть практически каждого приложе ния Win32. В них определяются такие объекты, как пиктограммы, кур соры, растровые изображения, таблицы строк, меню, диалоговые окна и многие другие. Ресурсы могут находиться, как в исполняемом (EXE) файле, так в библиотеках DLL.

Для поддержки ресурсов в проектах Visual C++ создается файл описания ресурсов, который обычно называется так же, как и проект, но имеет расширение .rc. Например, для проекта Sample файл описа ния ресурсов будет называться Sample.rc. Также вместе с файлом описания ресурсов создается заголовочный файл resource.h, содер жащий определения идентификаторов используемых ресурсов.

При создании нового ресурса или добавлении существующего ре сурса Visual Studio открывает соответствующий редактор ресурсов. Редакторы ресурсов содержат инструменты для быстрого и удобного изменения ресурсов приложения. В составе Visual Studio обязательно имеются редакторы для таких ресурсов, как пиктограммы, курсоры, растровые изображения, диалоговые окна, меню, таблица быстрых клавиш, таблица строк, панели инструментов и информация о версии приложения.

***Меню***

Меню является важнейшим элементом большинства оконных при ложений Windows. Меню расположенное ниже заголовка главного окна приложения называется *главным меню*. Окно, имеющее заголо вок, может предоставлять доступ к *системному меню*, которое вызы вается щелчком левой кнопки мыши на пиктограмме, расположенной в левой части главного окна.

Иногда в приложениях используются *контекстные меню*, появ ляющиеся под курсором при щелчке правой кнопкой мыши. Такие меню обычно ассоциируются с некоторым объектом, на который ука зывает курсор мыши в момент щелчка.

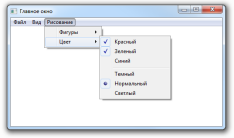
Любое меню содержит *пункты меню*. Пункт меню обозначается своим именем: словом или короткой фразой. Различают два типа пунк тов меню:

− *пункт-команда* – пункт меню, имеющий уникальный идентифика тор и который посылает приложению сообщение WM\_COMMAND, за ставляя его выполнить некоторое действие;

− *пункт-подменю* – пункт меню, представляет собой заголовок вы зываемого меню следующего, более низкого уровня.

Пункты меню могут быть *разрешенными* (enabled), *запрещенными* (disabled) или *недоступными* (grayed). По умолчанию пункт меню яв ляется разрешенным. Запрещенный и недоступный пункты с точки зрения поведения одинаковы. Различаются запрещенный и недоступ ный пункты только своим внешним видом: запрещенный выглядит также как и разрешенный, а недоступный отображается серым цветом.

Иногда пункт меню используется в роли *флажка* (check box), ко торый может быть установлен или сброшен. Пункты меню могут так же использоваться в роли *переключателей* (radio button). На рис. 1.8 показано окно, в котором вызвано меню, содержащее флажки и пере ключатели.

*Рис. 1.8.* Пункты меню флажки и переключатели

Меню можно создать одним из следующих способов:

− на основе шаблона меню, определенного в файле описания ресур сов;

− при помощи функций CreateMenu, AppendMenu и InsertMenuItem; − на основе шаблона меню, определяемого во время выполнения программы при помощи функции LoadMenuIndirect.

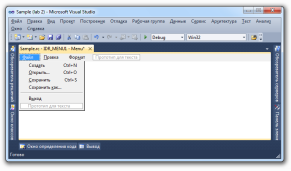
Чаще всего используют первый способ. Для того чтобы создать шаблон меню в файле описания ресурсов, нужно выполнить следую щие действия:

1. В окне **Обозреватель решений** (**Solution Explorer**) выберите нужный проект.

2. В меню **Проект** (**Project**) выберите **Добавить ресурс** (**Add Re source**).

Откроется диалоговое окно **Добавление ресурса** (**Add Resource**). 3. В области **Тип ресурса** (**Resource type**) выберите **Menu**. Нажмите кнопку **Создать** (**New**).

4. В открывшемся редакторе Visual Studio (рис. 1.9), отредактируйте и сохраните созданный шаблон меню.

*Рис. 1.9.* Шаблон меню в редакторе Visual Studio

После создания шаблона меню оно еще не появится в составе окна приложения. Чтобы это случилось, меню нужно присоединить к окну одним из нескольких способов.

Наиболее традиционным способом является присваивание полю *lpszMenuName* структуры WNDCLASSEX значения указателя на имя меню, при регистрации оконного класса. Если имя меню определено, как це лочисленная константа, нужно использовать макрос MAKEINTRESOURCE.

Можно связать меню с окном при создании окна, передав де скриптор меню параметру *hMenu* функции CreateWindowEx. В этом спо собе меню сначала нужно загрузить, используя функцию LoadMenu:

HMENU **LoadMenu**(HINSTANCE *hInstance*, LPCTSTR *lpMenuName*);

Первый параметр, *hInstance*, задает дескриптор модуля, который содержит шаблон меню.

Второй параметр, *lpMenuName*, определяет имя меню. Чтобы задать этот параметр, можно использовать макрос MAKEINTRESOURCE, в котором следует указать идентификатор шаблона меню.

При успешном завершении функция LoadMenu вернет дескриптор меню, в противном случае NULL.

Есть еще один способ назначения меню – использовать функцию SetMenu, которая имеет следующий прототип:

BOOL **SetMenu**(HWND *hWnd*, HMENU *hMenu*);

Первый параметр, *hWnd*, – дескриптор окна, к которому нужно при соединить меню. Второй параметр, *hMenu*, – дескриптор меню.

В случае успеха функция SetMenu возвращает TRUE, в противном случае FALSE. Новое меню заменяет старое меню, если оно уже было. В Win32 API имеется несколько функций (табл. 1.16), которые можно использовать для работы с меню. Подробное описание этих функций см. в документации Platform SDK.

*Таблица 1.16.* Функции для работы с меню

| **Функция** | **Описание** |
| --- | --- |
| AppendMenu | Добавляет новый пункт меню в конец меню |
| CreateMenu | Создает меню |
| DeleteMenu | Удаляет пункт меню |
| EnableMenuItem | Делает пункт меню разрешенным, запрещенным или недоступным |
| GetMenu | Возвращает дескриптор меню, связанного с указан ным окном |
| GetSubMenu | Возвращает дескриптор подменю |
| InsertMenuItem | Вставляет в меню новый пункт |
| RemoveMenu | Удаляет пункт меню |
| SetMenuDefaultItem | Делает пункт меню «применяемым по умолчанию» |
| SetMenuItemInfo | Изменяет информацию о пункте меню (в том числе об отметке пунктов-флажков и пунктов-переключателей) |
| TrackPopupMenuEx | Отображает контекстное меню в заданном месте |

***Таблица быстрых клавиш***

*Быстрая клавиша* (keyboard accelerator) – это клавиша или комби нация клавиш, которые при нажатии генерируют сообщение WM\_COM MAND или WM\_SYSCOMAND.

Обычно быстрые клавиши дублируют пункты меню, предоставляя пользователям альтернативный способ вызова команд (например, стандартное приложение «Блокнот» использует комбинацию Ctrl+N для создания нового текстового файла). Однако быстрые клавиши мо

гут также генерировать команды, которых нет в меню.

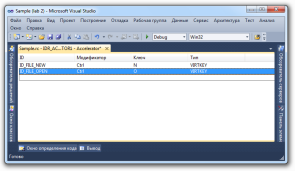
Чтобы добавить в приложение обработку быстрых клавиш, пер вым делом нужно создать таблицу быстрых клавиш. Для этого выпол ните следующие действия:

1. В окне **Обозреватель решений** (**Solution Explorer**) выберите нужный проект.

2. В меню **Проект** (**Project**) выберите **Добавить ресурс** (**Add Re source**).

Откроется диалоговое окно **Добавление ресурса** (**Add Resource**). 3. В области **Тип ресурса** (**Resource type**) выберите **Accelerator**. Нажмите кнопку **Создать** (**New**).

4. В открывшемся редакторе Visual Studio (рис. 1.10), отредактируйте и сохраните созданную таблицу быстрых клавиш.

*Рис. 1.10.* Таблица быстрых клавиш в редакторе Visual Studio

Когда таблица быстрых клавиш будет создана, необходимо обес печить загрузку этой таблицы во время запуска приложения. Для этого используется функция LoadAccelerators:

HACCEL **LoadAccelerators**(HINSTANCE *hInstance*,

LPCTSTR *lpTableName*);

Первый параметр, *hInstance*, задает дескриптор модуля, который содержит таблицу быстрых клавиш.

Второй параметр, *lpTableName*, определяет имя таблицы быстрых клавиш. Чтобы задать этот параметр, можно использовать макрос MAKEINTRESOURCE, в котором следует указать идентификатор ресурса таблицы быстрых клавиш.

При успешном завершении функция LoadAccelerators возвращает дескриптор, загруженной таблицы быстрых клавиш, в противном слу чае NULL.

Для обработки быстрых клавиш приложение должно перехваты вать сообщения клавиатуры, анализировать их коды и в случае совпа дения с кодом, определенным в таблице быстрых клавиш, направлять

соответствующее сообщение в оконную процедуру главного окна. Все это может выполнить функция TranslateAccelerator:

int **TranslateAccelerator**(HWND *hWnd*, HACCEL *hAccTable*, LPMSG *lpMsg*);

Первый параметр, *hWnd*, дескриптор окна, которому будут отправ ляться сообщения WM\_COMMAND или WM\_SYSCOMAND, если таблица быстрых клавиш содержит код нажатой виртуальной клавиши.

Второй параметр, *hAccTable*, дескриптор загруженной таблицы быстрых клавиш.

Последний параметр, *lpMsg*, указывает на сообщение, которое нужно обработать.

Если функция TranslateAccelerator, возвращается ненулевое зна чение, это значит, что преобразование комбинации клавиш и обработ ка сообщения завершились успешно. В этом случае приложение не должно повторно обрабатывать эту комбинацию при помощи функций TranslateMessage и DispatchMessage.

Данное требование можно выполнить, если организовать цикл об работки сообщений, например, следующим образом:

*Листинг 1.16.* Обработка быстрых клавиш

1 MSG msg;

2 BOOL bRet;

3

4 while ((bRet = GetMessage(&msg, NULL, 0, 0)) != FALSE) 5 {

6 if (!**TranslateAccelerator**(hWnd, hAccel, &msg)) 7 {

8 TranslateMessage(&msg);

9 DispatchMessage(&msg);

10 } // *if*

11 } // *while*

В этом примере hWnd – дескриптор главного окна приложения, а hAccel – дескриптор загруженной таблицы быстрых клавиш.

**Создание приложения Win32**

Для того чтобы создать приложение Win32 в Visual C++ необхо димо выполнить следующие действия:

1. В меню **Файл** (**File**) выберите **Создать** (**New**) → **Проект** (**Project**). Откроется диалоговое окно **Создать проект** (**New Project**).

2. В области шаблонов проектов **Visual C++** выберите группу **Win32** и затем выберите элемент **Проект Win32** (**Win32 Project**).

В поле **Имя** (**Name**) введите имя создаваемого проекта (например, SampleWin32). Нажмите кнопку **ОК**.

3. В открывшемся диалоговом окне **Мастер приложений Win32** (**Win32 Application Wizard**) нажмите кнопку **Далее** (**Next**).

Выберите **Приложение Windows** (**Windows application**) и пара метр **Пустой проект** (**Empty Project**), как показано на рис. 1.11.

Нажмите кнопку **Готово** (**Finish**).

*Рис. 1.11.* Создание приложения Win32 в Visual C++

4. В меню **Проект** (**Project**) выберите **Добавить ресурс** (**Add Re source**).

Откроется диалоговое окно **Добавление ресурса** (**Add Resource**). 5. В области **Тип ресурса** (**Resource type**) выберите **Dialog**. Нажмите кнопку **Создать** (**New**).

6. В открывшемся редакторе Visual Studio отредактируйте созданный шаблон диалогового окна, как показано на рис. 1.12.

7. В меню **Проект** (**Project**) выберите **Добавить ресурс** (**Add Re source**).

Откроется диалоговое окно **Добавление ресурса** (**Add Resource**).

109



*Рис. 1.12.* Шаблон диалогового окна

8. В области **Тип ресурса** (**Resource type**) выберите **Menu**. Нажмите кнопку **Создать** (**New**).

9. В открывшемся редакторе Visual Studio отредактируйте созданный шаблон меню, добавив в него пункты меню (табл. 1.17).

*Таблица 1.17.* Пункты меню

| **Идентификатор** | **Надпись** | **Пункт меню** |
| --- | --- | --- |
| ID\_NEW\_RECORD | &Добавить запись...\tCtrl+N | &Правка |
| ID\_NEW\_RECORD2 | Добавить &несколько записей... | &Правка |
| ID\_DEL\_RECORD | &Удалить запись\tDel | &Правка |
| ID\_FIND\_RECORD | Найти &запись\tCtrl+F | &Правка |
| ID\_FORMAT\_FONT | &Шрифт... | Форм&ат |

10. В меню **Проект** (**Project**) выберите **Добавить ресурс** (**Add Re source**).

Откроется диалоговое окно **Добавление ресурса** (**Add Resource**). 11. В области **Тип ресурса** (**Resource type**) выберите **Accelerator**. Нажмите кнопку **Создать** (**New**).

12. В открывшемся редакторе Visual Studio отредактируйте созданную таблицу быстрых клавиш, как показано на рис. 1.13.

*Рис. 1.13.* Таблица быстрых клавиш

13. Добавьте в проект исходный файл SampleWin32.cpp.

14. В файле, открывшемся в редакторе Visual Studio, введите про граммный код из примера в листинге 1.17 и сохраните этот файл.

15. Включите в проект файл манифеста.

16. В меню **Построение** (**Build**) выберите команду **Построить реше ние** (**Build Solution**).

*Листинг 1.17.* Файл исходного кода SampleWin32.cpp

1 #include <Windows.h>

2 #include <WindowsX.h>

3 #include <CommCtrl.h>

4 #include <tchar.h>

5

6 #include "resource.h"

7 #include "afxres.h"

8

9 // *идентификаторы элементов управления на главном окне* 10 #define IDC\_LIST 2001

11

12 // *код собственного сообщения WM\_ADDITEM,*

13 // *которое будет использоваться для добавления записей* 14 #define WM\_ADDITEM WM\_USER + 1

15

16 HWND hWnd = NULL; // *дискриптор главного окна*

17

18 HACCEL hAccel = NULL; // *дискриптор таблицы быстрых клавиш* 19

20 // *дескрипторы немодальных диалоговых окон*

21 HWND hDlg = NULL;

22 HWND hFindDlg = NULL;

23

24 TCHAR szBuffer[100] = TEXT("");

25

26 FINDREPLACE findDlg; // *структура для диалогового окна "Найти"* 27 UINT uFindMsgString = 0; // *код сообщения FINDMSGSTRING* 28

29 HFONT hFont = NULL; // *дескриптор шрифта*

30

31 // *оконная процедура главного окна*

32 LRESULT CALLBACK MyWindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

33

34 // *функция, которая вызывается в цикле обработки сообщений,* 35 // *пока в очереди нет сообщений*

36 void OnIdle(HWND hwnd);

37 // *функция, которая вызывается в цикле обработки сообщений*

38 // *перед тем, как сообщение будет передано в оконную процедуру* 39 BOOL PreTranslateMessage(LPMSG lpMsg);

40

41 // *далее идут обработчики сообщений главного окна:* 42 // *обработчик сообщения WM\_CREATE*

43 BOOL OnCreate(HWND hwnd, LPCREATESTRUCT lpCreateStruct); 44 // *обработчик сообщения WM\_DESTROY*

45 void OnDestroy(HWND hwnd);

46 // *обработчик сообщения WM\_SIZE*

47 void OnSize(HWND hwnd, UINT state, int cx, int cy); 48 // *обработчик сообщения WM\_COMMAND*

49 void OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify); 50 // *обработчик сообщения WM\_ADDITEM*

51 void OnAddItem(HWND hwnd);

52 // *обработчик сообщения FindMsgString*

53 void OnFindMsgString(HWND hwnd, LPFINDREPLACE lpFindReplace); 54

55 // *процедура диалогового окна*

56 INT\_PTR CALLBACK DialogProc(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM lParam);

57

58 // *далее идут обработчики сообщений диалогового окна:* 59 // *обработчик сообщения WM\_INITDIALOG*

60 BOOL Dialog\_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);

61 // *обработчик сообщения WM\_CLOSE*

62 void Dialog\_OnClose(HWND hwnd);

63 // *обработчик сообщения WM\_COMMAND*

64 void Dialog\_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify);

65

66 // -------------------------------------------------------------- 67 int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpszCmdLine, int nCmdShow)

68 {

69 // *регистрируем оконный класс главного окна...* 70

71 WNDCLASSEX wcex = { sizeof(WNDCLASSEX) };

72 wcex.style = CS\_HREDRAW|CS\_VREDRAW|CS\_DBLCLKS; 73 wcex.lpfnWndProc = MyWindowProc; // *оконная процедура* 74 wcex.hInstance = hInstance;

75 wcex.hIcon = LoadIcon(NULL, IDI\_APPLICATION); 76 wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);

77 wcex.hbrBackground = (HBRUSH)(COLOR\_BTNFACE + 1); 78 wcex.lpszMenuName = MAKEINTRESOURCE(IDR\_MENU1); 79 wcex.lpszClassName = TEXT("MyWindowClass"); // *имя класса*

80 wcex.hIconSm = LoadIcon(NULL, IDI\_APPLICATION); 81

82 if (0 == RegisterClassEx(&wcex)) // *регистрируем класс* 83 {

84 // *не удалось зарегистрировать новый оконный класс* 85 return -1; // *завершаем работу приложения* 86 } // *if*

87

88 // *загружаем библиотеку*

89 // *элементов управления общего пользования* 90 LoadLibrary(TEXT("ComCtl32.dll"));

91

92 // *загружаем таблицу быстрых клавиш*

93 hAccel = LoadAccelerators(hInstance, MAKEINTRESOURCE( IDR\_ACCELERATOR1));

94

95 // *создаем главное окно на основе нового оконного класса* 96 hWnd = CreateWindowEx(0, TEXT("MyWindowClass"), TEXT("SampleWin32"), WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

97

98 if (NULL == hWnd) // *если не удалось создать окно* 99 {

100 return -1; // *завершаем работу приложения* 101 } // *if*

102

103 ShowWindow(hWnd, nCmdShow); // *отображаем главное окно* 104

105 // *далее идет цикл обработки сообщения...* 106

107 MSG msg;

108 BOOL bRet;

109

110 for (;;)

111 {

112 // *определяем наличие сообщений в очереди* 113 while (!PeekMessage(&msg, NULL, 0, 0, PM\_NOREMOVE)) 114 {

115 OnIdle(hWnd);

116 } // *while*

117

118 // *извлекаем сообщение из очереди*

119 bRet = GetMessage(&msg, NULL, 0, 0);

120

121 if (bRet == -1)

122 {

123 /\* *обработка ошибки и возможно выход из цикла* \*/ 124 } // *if*

125 else if (FALSE == bRet)

126 {

127 break; // *получено WM\_QUIT, выход из цикла* 128 } // *if*

129 else if (!PreTranslateMessage(&msg))

130 {

131 TranslateMessage(&msg);

132 DispatchMessage(&msg);

133 } // *if*

134 } // *for*

135

136 return (int)msg.wParam; // *завершаем работу приложения* 137 } // *\_tWinMain*

138

139 // -------------------------------------------------------------- 140 LRESULT CALLBACK MyWindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

141 {

142 switch (uMsg)

143 {

144 HANDLE\_MSG(hWnd, WM\_CREATE, OnCreate);

145 HANDLE\_MSG(hWnd, WM\_DESTROY, OnDestroy); 146 HANDLE\_MSG(hWnd, WM\_SIZE, OnSize);

147 HANDLE\_MSG(hWnd, WM\_COMMAND, OnCommand); 148 case WM\_ADDITEM:

149 OnAddItem(hWnd);

150 return 0;

151 } // *switch*

152

153 if (uFindMsgString == uMsg) // *сообщение FINDMSGSTRING* 154 {

155 OnFindMsgString(hWnd, (LPFINDREPLACE)lParam); 156 return 0;

157 } // *if*

158

159 // *передача необработанного сообщения*

160 // *оконной процедуре по умолчанию*

161 return DefWindowProc(hWnd, uMsg, wParam, lParam); 162 } // *MyWindowProc*

163

164 // -------------------------------------------------------------- 165 void OnIdle(HWND hwnd)

166 {

167 /\* *здесь можно выполнять обработку каких-либо данных* \*/

168 } // *OnIdle*

169

170 // -------------------------------------------------------------- 171 BOOL PreTranslateMessage(LPMSG lpMsg)

172 {

173 BOOL bRet = TRUE;

174

175 if (!TranslateAccelerator(hWnd, hAccel, lpMsg)) 176 {

177 bRet = IsDialogMessage(hDlg, lpMsg);

178

179 if (FALSE == bRet)

180 bRet = IsDialogMessage(hFindDlg, lpMsg); 181 } // *if*

182

183 return bRet;

184 } // *PreTranslateMessage*

185

186 // -------------------------------------------------------------- 187 BOOL OnCreate(HWND hwnd, LPCREATESTRUCT lpCreateStruct) 188 {

189 // *создаем список*

190 CreateWindowEx(0, TEXT("ListBox"), NULL,

191 WS\_CHILD|WS\_VISIBLE|WS\_BORDER|LBS\_STANDARD, 10, 10, 250, 410, hwnd, (HMENU)IDC\_LIST, lpCreateStruct->hInstance, NULL); 192

193 // *создаем кнопку "Добавить запись"*

194 CreateWindowEx(0, TEXT("Button"), TEXT("Добавить запись"), 195 WS\_CHILD|WS\_VISIBLE|BS\_PUSHBUTTON, 270, 10, 200, 40, hwnd, (HMENU)ID\_NEW\_RECORD, lpCreateStruct->hInstance, NULL); 196

197 // *создаем кнопку "Добавить неск. записей"*

198 CreateWindowEx(0, TEXT("Button"), TEXT("Добавить неск. записей"),

199 WS\_CHILD|WS\_VISIBLE|BS\_PUSHBUTTON, 270, 55, 200, 40, hwnd, (HMENU)ID\_NEW\_RECORD2, lpCreateStruct->hInstance, NULL); 200

201 // *создаем кнопку "Удалить запись"*

202 CreateWindowEx(0, TEXT("Button"), TEXT("Удалить запись"), 203 WS\_CHILD|WS\_VISIBLE|BS\_PUSHBUTTON, 270, 100, 200, 40, hwnd, (HMENU)ID\_DEL\_RECORD, lpCreateStruct->hInstance, NULL); 204

205 // *создаем кнопку "Найти запись"*

206 CreateWindowEx(0, TEXT("Button"), TEXT("Найти запись"), 207 WS\_CHILD|WS\_VISIBLE|BS\_PUSHBUTTON, 270, 145, 200, 40, hwnd, (HMENU)ID\_FIND\_RECORD, lpCreateStruct->hInstance, NULL);

208

209 // *создаем кнопку "Изменить шрифт"*

210 CreateWindowEx(0, TEXT("Button"), TEXT("Изменить шрифт"), 211 WS\_CHILD|WS\_VISIBLE|BS\_PUSHBUTTON, 270, 210, 200, 40, hwnd, (HMENU)ID\_FORMAT\_FONT, lpCreateStruct->hInstance, NULL); 212

213 return TRUE;

214 } // *OnCreate*

215

216 // -------------------------------------------------------------- 217 void OnDestroy(HWND hwnd)

218 {

219 // *удаляем созданный шрифт*

220 if (NULL != hFont)

221 DeleteObject(hFont), hFont = NULL;

222

223 PostQuitMessage(0); // *отправляем сообщение WM\_QUIT* 224 } // *OnDestroy*

225

226 // -------------------------------------------------------------- 227 void OnSize(HWND hwnd, UINT state, int cx, int cy) 228 {

229 if (state != SIZE\_MINIMIZED)

230 {

231 // *получим дескриптор списка*

232 HWND hwndCtl = GetDlgItem(hwnd, IDC\_LIST); 233 // *изменяем высоту списка*

234 MoveWindow(hwndCtl, 10, 10, 250, cy-20, TRUE); 235 } // *if*

236 } // *OnSize*

237

238 // -------------------------------------------------------------- 239 void OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify) 240 {

241 // *получим дескриптор экземпляра приложения*

242 HINSTANCE hInstance = GetWindowInstance(hwnd); 243

244 switch (id)

245 {

246 case ID\_NEW\_RECORD: // *нажата кнопка "Добавить запись"* 247 {

248 // *создаем модальное диалоговое окно* 249 int nDlgResult = DialogBox(hInstance, MAKEINTRESOURCE(IDD\_DIALOG1), hwnd, DialogProc);

250

251 if (IDOK == nDlgResult)

252 {

253 // *отправляем окну сообщение о том, что нужно добавить запись*

254 SendMessage(hwnd, WM\_ADDITEM, 0 ,0); 255 } // *if*

256 }

257 break;

258

259 case ID\_NEW\_RECORD2: // *нажата кнопка "Добавить неск. записей"*

260 // *если немодальное диалоговое окно еще не создано* 261 if (IsWindow(hDlg) == FALSE)

262 {

263 // *создаем немодальное диалоговое окно* 264 hDlg = CreateDialog(hInstance, MAKEINTRESOURCE( IDD\_DIALOG1), hwnd, DialogProc);

265 // *отображаем немодальное диалоговое окно* 266 ShowWindow(hDlg, SW\_SHOW);

267 } // *if*

268 break;

269

270 case ID\_DEL\_RECORD: // *нажата кнопка "Удалить запись"* 271 {

272 // *получим дескриптор списка*

273 HWND hwndCtl = GetDlgItem(hwnd, IDC\_LIST); 274

275 // *определим текущий выделенный элемент в списке* 276 int iItem = ListBox\_GetCurSel(hwndCtl); 277

278 if (iItem != -1)

279 {

280 int mbResult = MessageBox(hwnd, TEXT("Удалить выбранный элемент?"), TEXT("SampleWin32"),

MB\_YESNO|MB\_ICONQUESTION);

281

282 if (mbResult == IDYES)

283 {

284 // *удаляем выделенный элемент из списка* 285 ListBox\_DeleteString(hwndCtl, iItem); 286 } // *if*

287 } // *if*

288 }

289 break;

290

291 case ID\_FIND\_RECORD: // *нажата кнопка "Найти запись"* 292 if (0 == uFindMsgString)

293 {

294 // *получим код сообщения FINDMSGSTRING* 295 uFindMsgString = RegisterWindowMessage( FINDMSGSTRING);

296 } // *if*

297

298 // *если диалоговое окно "Найти" еще не создано* 299 if (IsWindow(hFindDlg) == FALSE)

300 {

301 findDlg.lStructSize = sizeof(FINDREPLACE); 302

303 // *указываем дескриптор экземпляра приложения* 304 findDlg.hInstance = hInstance;

305 // *указываем дескриптор окна владельца* 306 findDlg.hwndOwner = hwnd;

307

308 // *указываем строковый буфер*

309 findDlg.lpstrFindWhat = szBuffer;

310 // *указываем размер буфера*

311 findDlg.wFindWhatLen = \_countof(szBuffer); 312

313 // *создаем диалоговое окно "Найти"*

314 hFindDlg = FindText(&findDlg);

315 } // *if*

316 break;

317

318 case ID\_FORMAT\_FONT: // *нажата кнопка "Изменить шрифт"* 319 {

320 CHOOSEFONT cf = { sizeof(CHOOSEFONT) }; 321

322 // *указываем дескриптор экземпляра приложения* 323 cf.hInstance = hInstance;

324 // *указываем дескриптор окна владельца* 325 cf.hwndOwner = hwnd;

326

327 LOGFONT lf;

328 ZeroMemory(&lf, sizeof(lf));

329

330 cf.lpLogFont = &lf; // *указываем структуру, которая будет использоваться для создания шрифта*

331

332 BOOL bRet = ChooseFont(&cf);

333

334 if (FALSE != bRet)

335 {

336 // *создаем новый шрифт*

337 HFONT hNewFont = CreateFontIndirect( cf.lpLogFont);

338

339 if (NULL != hNewFont)

340 {

341 // *удаляем созданный ранее шрифт* 342 if (NULL != hFont) DeleteObject(hFont); 343 // *устанавливаем новый шрифт для списка* 344 hFont = hNewFont;

345 SendDlgItemMessage(hwnd, IDC\_LIST, WM\_SETFONT, (WPARAM)hFont, (LPARAM)TRUE);

346 } // *if*

347 } // *if*

348 }

349 break;

350 } // *switch*

351 } // *OnCommand*

352

353 // -------------------------------------------------------------- 354 void OnAddItem(HWND hwnd)

355 {

356 // *получим дескриптор списка*

357 HWND hwndCtl = GetDlgItem(hwnd, IDC\_LIST);

358

359 // *добавляем новый элемент в список*

360 int iItem = ListBox\_AddString(hwndCtl, szBuffer); 361

362 // *выделяем новый элемент*

363 ListBox\_SetCurSel(hwndCtl, iItem);

364 } // *OnAddItem*

365

366 // -------------------------------------------------------------- 367 void OnFindMsgString(HWND hwnd, LPFINDREPLACE lpFindReplace) 368 {

369 if (lpFindReplace->Flags & FR\_FINDNEXT) // *нажата кнопка "Найти далее"*

370 {

371 // *получим дескриптор списка*

372 HWND hwndCtl = GetDlgItem(hwnd, IDC\_LIST); 373

374 // *определим текущий выделенный элемент в списке* 375 int iItem = ListBox\_GetCurSel(hwndCtl);

376

377 // *выполним поиск указанного текста в списке* 378 // *сразу после текущего выделенного элемента*

379 iItem = ListBox\_FindString(hwndCtl, iItem, lpFindReplace- >lpstrFindWhat);

380

381 // *выделяем найденный элемент*

382 ListBox\_SetCurSel(hwndCtl, iItem);

383

384 if (LB\_ERR == iItem) // *элемент не найден* 385 {

386 MessageBox(hFindDlg, TEXT("Поиск завершен"), TEXT("SampleWin32"), MB\_OK|MB\_ICONINFORMATION);

387 } // *if*

388 } // *if*

389 } // *OnFindMsgString*

390

391 // -------------------------------------------------------------- 392 INT\_PTR CALLBACK DialogProc(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)

393 {

394 switch (uMsg)

395 {

396 case WM\_INITDIALOG:

397 {

398 BOOL bRet = HANDLE\_WM\_INITDIALOG(hwndDlg, wParam, lParam, Dialog\_OnInitDialog);

399 return SetDlgMsgResult(hwndDlg, uMsg, bRet); 400 }

401 case WM\_CLOSE:

402 HANDLE\_WM\_CLOSE(hwndDlg, wParam, lParam, Dialog\_OnClose); 403 return TRUE;

404 case WM\_COMMAND:

405 HANDLE\_WM\_COMMAND(hwndDlg, wParam, lParam, Dialog\_OnCommand);

406 return TRUE;

407 } // *switch*

408

409 return FALSE;

410 } // *DialogProc*

411

412 // -------------------------------------------------------------- 413 BOOL Dialog\_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam)

414 {

415 // *получим дескриптор окна редактируемого поля* 416 HWND hwndEdit = GetDlgItem(hwnd, IDC\_EDIT1); 417

418 // *задаем максимальную длину текста в редактируемом поле*

419 Edit\_LimitText(hwndEdit, \_countof(szBuffer)-1); 420

421 // *задаем серый (фоновый) текст в редактируемом поле* 422 Edit\_SetCueBannerText(hwndEdit, L"Название новой записи"); 423

424 return TRUE;

425 } // *Dialog\_OnInitDialog*

426

427 // -------------------------------------------------------------- 428 void Dialog\_OnClose(HWND hwnd)

429 {

430 if (hwnd == hDlg)

431 {

432 // *уничтожаем немодальное диалоговое окно* 433 DestroyWindow(hwnd);

434 } // *if*

435 else

436 {

437 // *завершаем работу модального диалогового окна* 438 EndDialog(hwnd, IDCLOSE);

439 } // *else*

440 } // *Dialog\_OnClose*

441

442 // -------------------------------------------------------------- 443 void Dialog\_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)

444 {

445 switch (id)

446 {

447 case IDOK: // *нажата кнопка "ОК"*

448 {

449 // *получим содержимое редактируемого поля* 450 int cch = GetDlgItemText(hwnd, IDC\_EDIT1, szBuffer, \_countof(szBuffer));

451

452 if (0 == cch) // *в редактируемого поле нет текста* 453 {

454 // *получим дескриптор окна редактируемого поля* 455 HWND hwndEdit = GetDlgItem(hwnd, IDC\_EDIT1); 456

457 EDITBALLOONTIP ebt = { sizeof(EDITBALLOONTIP) }; 458

459 ebt.pszTitle = L"SampleWin32";

460 ebt.pszText = L"Укажите название новой записи"; 461 ebt.ttiIcon = TTI\_WARNING;

462

463 Edit\_ShowBalloonTip(hwndEdit, &ebt); 464 } // *if*

465 else if (hwnd == hDlg)

466 {

467 // *очистим редактируемое поле*

468 SetDlgItemText(hwnd, IDC\_EDIT1, NULL); 469 // *отправляем окну-владельцу сообщение о том, что нужно добавить запись*

470 SendMessage(GetParent(hwnd), WM\_ADDITEM, 0, 0); 471 } // *if*

472 else

473 {

474 // *завершаем работу модального диалогового окна* 475 EndDialog(hwnd, IDOK);

476 } // *else*

477 } // *if*

478 break;

479

480 case IDCANCEL: // *нажата кнопка "Отмена"*

481 if (hwnd == hDlg)

482 {

483 // *уничтожаем немодальное диалоговое окно* 484 DestroyWindow(hwnd);

485 } // *if*

486 else

487 {

488 // *завершаем работу модального диалогового окна* 489 EndDialog(hwnd, IDCANCEL);

490 } // *else*

491 break;

492 } // *switch*

493 } // *Dialog\_OnCommand*

**Задание к работе**

1. Изучить элементы управления общего пользования, указанные в варианте задания. Особое внимание уделить тому, какие стили они поддерживают, какие сообщения принимают и какие уведомления отправляют родительскому окну. Включить в отчет назначение и описание изученных элементов управления.

2. Изучить диалоговые окна общего пользования, указанные в вари анте задания. Включить в отчет назначение и описание изученных диалоговых окон.

Изучить оконное сообщение WM\_TIMER и сообщения, указанные в 3.

варианте задания. Включить в отчет описание изученных оконных сообщений.

4. Разработать в Visual C++ оконное приложение Win32, которое:

− должно создавать главное окно, содержащее меню и элементы управления, указанные в варианте задания;

− должно обрабатывать комбинации быстрых клавиш; − должно создавать одно или несколько диалоговых окон с ис пользованием шаблона;

− должно создавать диалоговое окно сообщений;

− должно создавать диалоговые окна, указанные в варианте за дания;

− должно обрабатывать оконное сообщение WM\_TIMER и оконные сообщения, указанные в варианте задания.

*При необходимости можно также использовать и другие эле менты управления.*

5. Протестировать работу приложения, разработанного в п. 4, на компьютере под управлением Windows XP (или выше). Результаты тестирования отразить в отчете.

6. Включить в отчет исходный программный код и выводы о проде ланной работе.

**Варианты заданий**

| **№** | **Элементы управления** | **Диалоговое окно** | **Сообщения** |
| --- | --- | --- | --- |
| 1,16 | Выпадающий список (Combo box)  Гиперссылка (SysLink)  Кнопка (Button)  Поле ввода IP-адреса (IP ad dress) | Цвет (Color)  Найти (Find) | WM\_CHAR  WM\_KEYDOWN  WM\_LBUTTONDBLCLK WM\_LBUTTONDOWN  WM\_LBUTTONUP  WM\_MOUSEMOVE  WM\_MOVE |
| 2,17 | Групповая рамка (Group box) Дерево просмотра (Tree view) Надпись (Static text)  Поле ввода горячей клавиши (Hot key) | Шрифт (Font)  Заменить (Replace) | WM\_CHAR  WM\_KEYUP  WM\_MBUTTONDBLCLK WM\_MBUTTONDOWN  WM\_MBUTTONUP  WM\_MOUSEWHEEL  WM\_MOVING |

| **№** | **Элементы управления** | **Диалоговое окно** | **Сообщения** |
| --- | --- | --- | --- |
| 3,18 | Закладки (Tab control)  Индикатор процесса (Progress bar)  Переключатель (Radio button) Поле ввода даты или времени (Data time picker) | Открыть (Open) Найти (Find) | WM\_MOUSEMOVE  WM\_RBUTTONDBLCLK WM\_RBUTTONDOWN  WM\_RBUTTONUP  WM\_SIZE  WM\_SYSCHAR  WM\_SYSKEYDOWN |
| 4,19 | Календарь (Month calendar) Редактируемое поле (Edit box)  Список (List box)  Счетчик или стрелки (Spin) | Сохранить как  (Save As)  Заменить (Replace) | WM\_LBUTTONDBLCLK WM\_LBUTTONDOWN  WM\_LBUTTONUP  WM\_MOUSEWHEEL  WM\_SIZING  WM\_SYSCHAR  WM\_SYSKEYUP |
| 5,20 | Кнопка (Button)  Поле ввода IP-адреса (IP ad dress)  Регулятор (Slider)  Список просмотра (List view) | Печать (Print)  Найти (Find) | WM\_CHAR  WM\_KEYDOWN  WM\_MBUTTONDBLCLK WM\_MBUTTONDOWN  WM\_MBUTTONUP  WM\_MOUSEMOVE  WM\_MOVE |
| 6,21 | Выпадающий список (Combo box)  Надпись (Static text)  Поле ввода горячей клавиши (Hot key)  Флажок (Check box) | Цвет (Color)  Заменить (Replace) | WM\_CHAR  WM\_KEYUP  WM\_MOUSEWHEEL  WM\_MOVING  WM\_RBUTTONDBLCLK WM\_RBUTTONDOWN  WM\_RBUTTONUP |
| 7,22 | Гиперссылка (SysLink)  Дерево просмотра (Tree view) Переключатель (Radio button) Поле ввода даты или времени (Data time picker) | Шрифт (Font)  Найти (Find) | WM\_LBUTTONDBLCLK WM\_LBUTTONDOWN  WM\_LBUTTONUP  WM\_MOUSEMOVE  WM\_SIZE  WM\_SYSCHAR  WM\_SYSKEYDOWN |
| 8,23 | Закладки (Tab control)  Индикатор процесса (Progress bar)  Редактируемое поле (Edit box)  Счетчик или стрелки (Spin) | Открыть (Open) Заменить (Replace) | WM\_MBUTTONDBLCLK WM\_MBUTTONDOWN  WM\_MBUTTONUP  WM\_MOUSEWHEEL  WM\_SIZING  WM\_SYSCHAR  WM\_SYSKEYUP |

| **№** | **Элементы управления** | **Диалоговое окно** | **Сообщения** |
| --- | --- | --- | --- |
| 9,24 | Календарь (Month calendar) Поле ввода IP-адреса (IP ad dress)  Регулятор (Slider)  Список (List box) | Сохранить как  (Save As)  Найти (Find) | WM\_CHAR  WM\_KEYDOWN  WM\_MOUSEMOVE  WM\_MOVE  WM\_RBUTTONDBLCLK WM\_RBUTTONDOWN  WM\_RBUTTONUP |
| 10,25 | Поле ввода горячей клавиши (Hot key)  Список просмотра (List view) Счетчик или стрелки (Spin) Флажок (Check box) | Печать (Print)  Заменить (Replace) | WM\_CHAR  WM\_KEYUP  WM\_LBUTTONDBLCLK WM\_LBUTTONDOWN  WM\_LBUTTONUP  WM\_MOUSEWHEEL  WM\_MOVING |
| 11,26 | Выпадающий список (Combo box)  Кнопка (Button)  Поле ввода даты или времени (Data time picker)  Регулятор (Slider) | Цвет (Color)  Найти (Find) | WM\_MBUTTONDBLCLK WM\_MBUTTONDOWN  WM\_MBUTTONUP  WM\_MOUSEMOVE  WM\_SIZE  WM\_SYSCHAR  WM\_SYSKEYDOWN |
| 12,27 | Гиперссылка (SysLink)  Дерево просмотра (Tree view) Индикатор процесса (Progress bar)  Редактируемое поле (Edit box) | Шрифт (Font)  Заменить (Replace) | WM\_MOUSEWHEEL  WM\_RBUTTONDBLCLK WM\_RBUTTONDOWN  WM\_RBUTTONUP  WM\_SIZING  WM\_SYSCHAR  WM\_SYSKEYUP |
| 13,28 | Групповая рамка (Group box) Закладки (Tab control)  Поле ввода IP-адреса (IP ad dress)  Регулятор (Slider) | Открыть (Open) Найти (Find) | WM\_CHAR  WM\_KEYDOWN  WM\_LBUTTONDBLCLK WM\_LBUTTONDOWN  WM\_LBUTTONUP  WM\_MOUSEMOVE  WM\_MOVE |
| 14,29 | Индикатор процесса (Progress bar)  Кнопка (Button)  Поле ввода горячей клавиши (Hot key)  Список (List box) | Сохранить как  (Save As)  Заменить (Replace) | WM\_CHAR  WM\_KEYUP  WM\_MBUTTONDBLCLK WM\_MBUTTONDOWN  WM\_MBUTTONUP  WM\_MOUSEWHEEL  WM\_MOVING |

| **№** | **Элементы управления** | **Диалоговое окно** | **Сообщения** |
| --- | --- | --- | --- |
| 15,30 | Календарь (Month calendar) Поле ввода даты или времени (Data time picker)  Список просмотра (List view) Флажок (Check box) | Печать (Print)  Найти (Find) | WM\_MOUSEMOVE  WM\_RBUTTONDBLCLK WM\_RBUTTONDOWN  WM\_RBUTTONUP  WM\_SIZE  WM\_SYSCHAR  WM\_SYSKEYDOWN |

**Контрольные вопросы**

1. Что в Windows называют окном?

2. Что такое главное окно приложения Windows? Сколько таких окон может быть у приложения?

3. Что в Windows называют диалоговое окном? Какие различают ви ды диалоговых окон?

4. Что в Windows называют элементами управления? 5. Что такое клиентская и не клиентская область окна? Что они в себя включают?

6. Какие системы координат используются в функциях Win32 API, работающих с окнами?

7. Какие функции Win32 API следует использовать для преобразова ния из одной системы координат в другую?

8. Что такое оконный класс? Как создать новый оконный класс? 9. Что такое предопределенные оконные классы элементов управле ния общего пользования? Какие предопределенные оконные клас сы существуют?

10. Какие функции Win32 API следует использовать для того, чтобы создать окно?

11. Какие функции Win32 API следует использовать для создания раз личных диалоговых окон?

12. Какие функции Win32 API возвращают дескриптор окна? 13. Какие функции Win32 API изменяют отображение окна? 14. Какие функции Win32 API возвращают различные характеристики окна или его оконного класса?

15. Что в Windows называют оконным сообщением?

16. Что такое очередь сообщений?

17. Что такое цикл обработки сообщений? Как обрабатываются сооб щения диалоговых окон?

18. Что такое оконная процедура? Для чего применяется оконная про цедура?

19. Что такое оконная процедура диалогового окна? Чем она отлича ется от обычной оконной процедуры?

20. Какие оконные сообщения используются наиболее часто? 21. Чем отличаются оконные сообщения WM\_CREATE и WM\_INITDIALOG? 22. Чем отличаются оконные сообщения WM\_DESTROY и WM\_CLOSE? 23. Чем отличаются оконные сообщения WM\_COMMAND и WM\_NOTIFY? 24. В чем разница между синхронными и асинхронными оконными сообщениями?

25. Какую функцию Win32 API следует использовать для того, чтобы отправить синхронное оконное сообщение?

26. Какую функцию Win32 API следует использовать для того, чтобы отправить асинхронное оконное сообщение?

27. Зачем отправлять оконные сообщения элементам управления? Ка кие функции Win32 API следует для этого использовать? 28. Какие существуют макросы для отправки оконных сообщений элементам управления?

29. Какую функцию Win32 API следует использовать для того, чтобы отправить сообщение WM\_QUIT? Почему для этого нельзя использо вать другие функции отправки оконных сообщений?

30. Как и для чего применяют распаковщики сообщений? 31. Что такое приложение Win32? Как в Visual C++ создать проект приложения Win32?

32. Для чего в приложении Win32 предназначена функция WinMain? Какие существуют варианты функции WinMain?

33. Что такое точка входа в приложение Win32?

34. Что такое и для чего применяется манифест приложения? 35. Что такое ресурсы приложения?

36. Как создать главное меню приложения?

37. Как обрабатываются комбинации быстрых клавиш?