Лабораторная работа № 5. Использование механизма обмена сообщениями для управления окнами в ОС Windows

Описание функций WinAPI по управлению окнами и их стилями

Программы в ОС Windows управляются сообщениями. Все действия пользователей перехватываются системой и преобразуются в сообщения, направляемые программе, которая владеет окнами, к которым обращены действия пользователя. У каждой программы, состоящей хотя бы из одного потока, есть несколько собственных очередей сообщений: очередь асинхронных сообщений, очередь синхронных сообщений, очередь ответных сообщений и очередь виртуального ввода, куда и направляются все сообщения, касающиеся действий с окнами. В каждой программе есть главный цикл, который состоит из получения следующего сообщения и его обработки с помощью внутренней процедуры, соответствующей данному типу сообщений. Иногда некоторые сообщения вызывает эти процедуры, минуя очереди сообщений.

Практически каждая программа в ОС Windows будет содержать код, похожий на приведенный в следующем листинге.

```
#include <windows.h>

//  Прототип для главной оконной процедуры
LRESULT CALLBACK WindowProcedure(HWND, UINT, WPARAM, LPARAM);

//  Имя класса окна делается глобальной переменной
char g_szClassName[] = "WindowApplication";

int WINAPI WinMain(HINSTANCE ThisInstance
                   , HINSTANCE PreviousInstance
                   , LPSTR lpszArgument
                   , int CommandShow
                  )
{
    HWND FirstWindow;       // Дескриптор окна
    MSG  Messages;          // Здесь сохраняются все сообщения в приложении
    WNDCLASS WindowClass;   // Структура данных для пользовательского класса окна

    WindowClass.hInstance = ThisInstance;
    WindowClass.lpszClassName = g_szClassName;

    // указатель на главную оконную процедуру
    WindowClass.lpfnWndProc = WindowProcedure;

    WindowClass.lpszMenuName = NULL;           // Нет системного меню

    // использование иконки по умолчанию
    WindowClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);

    WindowClass.hCursor = LoadCursor(NULL, IDC_ARROW); // курсор мыши - стрелка

    // перерисовка всего окна при изменении высоты и ширины окна,
```

```
    // а также перехват двойного клика
    WindowClass.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;

    // нет дополнительных байтов для класса окна
    WindowClass.cbClsExtra = 0;

    // нет дополнительных байтов для экземпляра окна
    WindowClass.cbWndExtra = 0;

    // для фона используется цвет по умолчанию
    WindowClass.hbrBackground = (HBRUSH)COLOR_BACKGROUND;

    // зарегистрировать класс окна, а в случае ошибки завершить программу
    if (0 == RegisterClass(&WindowClass))
        return EXIT_FAILURE;

    // создание окна
    FirstWindow = CreateWindow(g_szClassName              // Имя класса
                               , "Приложение с окошком"   // заголовок окна
                               , WS_OVERLAPPEDWINDOW      // перекрываемое окно
                               , CW_USEDEFAULT            // ОС решает за нас,
                               , CW_USEDEFAULT            // куда разместить окно
                               , 600                      // ширина окна
                               , 375                      // высота окна
                               , HWND_DESKTOP  // это окно - потомок рабочего стола
                               , NULL          // меню не выводится
                               , ThisInstance  // дескриптор экземпляра программы
                               , NULL          // без данных для создания окна
                              );
    // Вывести (показать) окно на экран
    ShowWindow(FirstWindow, CommandShow);
    UpdateWindow(FirstWindow);

    // Цикл обработки сообщений
    while (GetMessage(&Messages, NULL, 0, 0))
    {
        // трансляция сообщений виртуальных клавиш в символьные сообщения
        TranslateMessage(&Messages);
        // отправка сообщений в оконную процедуру
        DispatchMessage(&Messages);
    }

    // Возвратить значение 0 из функции WinMain, которое она получит
    // транзитивно от PostQuitMessage()
    return Messages.wParam;
}

// Эта функция (главная оконная процедура) вызывается API-функцией DispatchMessage()
//
// FUNCTION: LRESULT CALLBACK WindowProcedure(HWND, UINT, WPARAM, LPARAM)
//
// PARAMETERS: [in]  hWindow          - хэндл окна
//             [in]  uMessage         - сообщение для окна
//             [in]  wFirstParameter  - дополнительная информация для сообщения
//             [in]  lSecondParameter - дополнительная информация для сообщения
//
// RETURN VALUE: 0   - во всех случаях
```

```
//
// COMMENTS:  Значение параметров wFirstParameter и lSecondParameter зависит от
//            параметра uMessage
//
LRESULT CALLBACK WindowProcedure(HWND hWindow
                                 , UINT uMessage
                                 , WPARAM wFirstParameter
                                 , LPARAM lSecondParameter
                                )
{
    switch (uMessage)                  // Обработка сообщений
    {
        case WM_CLOSE:     // закрытие окна
            DestroyWindow(hWindow);
            return 0;
        case WM_DESTROY:   // удаление окна
            PostQuitMessage(0);  // отправка сообщения WM_QUIT в очередь сообщений
            break;
        default:          // Для сообщений, с которыми мы не хотим связываться
            return DefWindowProc(hWindow
                                 , Message
                                 , wFirstParameter
                                 , lSecondParameter
                                );
    }
    return 0;
}
```

В ОС Windows прежде чем создавать окно, нужно зарегистрировать его класс. Регистрацию класса окна и производит следующая WinAPI-функция.

```
#include <windows.h>
// Функция регистрации класса окна
//
// FUNCTION: ATOM RegisterClass(CONST WNDCLASS*)
//
// PARAMETERS:  [in] lpWndClass   - указатель на структуру с данными класса
//
// RETURN VALUE: идентификатор класса окна - в случае успеха
//               0                         - в случае неудачи
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//            нужно вызвать GetLastError
//
ATOM RegisterClass(CONST WNDCLASS* lpWndClass);
```

Можно использовать следующие «системные» стили окон:

- *WS_BORDER* — Создание окна с рамкой.
- *WS_CAPTION* — Создание окна с заголовком (невозможно использовать одновременно со стилем *WS_DLGFRAME*).
- *WS_CHILD*
- *WS_CHILDWINDOW* — Создание дочернего окна (невозможно использовать одновременно со стилем *WS_POPUP*).
- *WS_CLIPCHILDREN* — Исключает область, занятую дочерним окном, при

выводе в родительское окно.

- *WS_CLIPSIBLINGS* — Используется совместно со стилем *WS_CHILD* для отрисовки в дочернем окне областей, перекрываемых другими окнами.
- *WS_DISABLED* — Создает окно, которое недоступно.
- *WS_DLGFRAME* — Создает окно с двойной рамкой, без заголовка.
- *WS_GROUP* — Позволяет объединять элементы управления в группы.
- *WS_HSCROLL* — Создает окно с горизонтальной полосой прокрутки.
- *WS_MAXIMIZE* — Создает окно максимального размера.
- *WS_MAXIMIZEBOX* — Создает окно с кнопкой развертывания окна.
- *WS_MINIMIZE*
- *WS_ICONIC* — Создает первоначально свернутое окно (используется только со стилем *WS_OWERLAPPED*).
- *WS_MINIMIZEBOX* — Создает окно с кнопкой свертывания.
- *WS_OVERLAPPED* — Создает перекрывающееся окно (которое, как правило, имеет заголовок и *WS_TILED* рамку).
- *WS_OVERLAPPEDWINDOW* — Создает перекрывающееся окно, имеющее стили *WS_OVERLAPPED*, *WS_CAPTION*, *WS_SYSMENU*, *WS_THICKFRAME*, *WS_MINIMIZEBOX*, *WS_MAXIMIZEBOX*.
- *WS_POPUP* — Создает всплывающее окно (невозможно использовать совместно со стилем *WS_CHILD*).
- *WS_POPUPWINDOW* — Создает всплывающее окно, имеющее стили *WS_BORDER*, *WS_POPUP*, *WS_SYSMENU*.
- *WS_SYSMENU* — Создает окно с кнопкой системного меню (можно использовать только с окнами, имеющими строку заголовка).
- *WS_TABSTOP* — Определяет элементы управления, переход к которым может быть выполнен по клавише *TAB*.
- *WS_THICKFRAME* — Создает окно с рамкой, используемой для изменения
- *WS_SIZEBOX* — размера окна.
- *WS_VISIBLE* - Создает первоначально неотображаемое окно.
- *WS_VSCROLL* — Создает окно с вертикальной полосой прокрутки.

Функция *CreateWindow* используется программой для создания окна, а пример ее использования приводился выше.

```
#include <windows.h>

// Функция создания окна
//
// FUNCTION: HWND CreateWindow(LPCTSTR, LPCTSTR, DWORD, LPARAM
//                              , int, int, int, int
//                              , HWND, HMENU, HINSTANCE, LPVOID
//                             )
//
```

```
// PARAMETERS: [in]  lpClassName   - указатель на зарегистрированное имя класса
//             [in]  lpWindowName  - указатель на имя окна
//             [in]  dwStyle       - стиль окна
//             [in]  x             - горизонтальная позиция окна
//             [in]  y             - вертикальная позиция окна
//             [in]  nWidth        - ширина окна
//             [in]  nHeight       - высота окна
//             [in]  hWndParent    - дескриптор родительского или окна-владельца
//             [in]  hMenu         - дескриптор меню или идентификатор дочернего окна
//             [in]  hInstance     - дескриптор экземпляра программы
//             [in]  lpParam       - указатель на данные создания окна
//
// RETURN VALUE: указатель на хэндл созданного окна - в случае успеха
//               NULL                               - в случае неудачи
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//            нужно вызвать GetLastError
//
HWND CreateWindow(LPCTSTR lpClassName, LPCTSTR lpWindowName, DWORD dwStyle
                  , int x, int y, int nWidth, int nHeight
                  , HWND hWndParent, HMENU hMenu, HANDLE hInstance, LPVOID lpParam
                 );
```

Следующие предопределенные классы элементов управления могут быть определены в параметре *ClassName.*

Предопределенные классы элементов управления пользовательским интерфейсом

BUTTON (КНОПКА). Обозначает маленькое прямоугольное дочернее окно, которое представляет собой кнопку и пользователь может щелкать по ней мышью, чтобы включить или отключить ее. Кнопки управления могут использоваться самостоятельно или в группах, и они могут или быть подписаны или появляться без текста. Кнопки управления обычно изменяют свой вид, когда пользователь щелкает мышью по ним.

COMBOBOX (*КОМБИНИРОВАННОЕ ОКНО*). Обозначает элемент управления, состоящий из окна со списком и поля выбора, похожего на элемент редактирования текста. При использовании этого стиля, прикладная программа должна отображать все время или окно со списком или включать раскрывающийся список. В зависимости от стиля комбинированного окна, пользователь сможет или не сможет редактировать содержание поля выбора. Если окно со списком видимо, вводимые символы внутри поля выбора подсвечивают сначала ввод в окно со списком, который соответствует печатаемым символам. Наоборот, выбор элемента в окне со списком отображает выбранный текст в поле выбора.

EDIT (*ОКНО РЕДАКТИРОВАНИЯ*). Обозначает прямоугольное дочернее окно, внутри которого пользователь может напечатать с клавиатуры текст. Пользователь выбирает элемент управления и дает ему фокус клавиатуры, щелкая мышью по нему или перемещаясь в него, путем нажатия клавиши табуляции *(TAB).* Пользователь

может напечатать текст, когда элемент управления редактируемого окна отображает мигающую каретку *(caret);* использовать мышь, чтобы перемещать курсор, выбирать символы, которые будут заменены, или установить курсор для вставки символов; или использовать *BACKSPACE*, чтобы удалять символы. Элементы управления редактируемого окна используют шрифт системы с переменным шагом и показывают на экране символы из символьного набора ANSI. Сообщение *WM_SETFONT* может также быть послано элементу управления окна редактирования, чтобы изменить заданный по умолчанию шрифт. Элементы управления окна редактирования увеличиваются в соответствии с табуляцией символов стольким количеством пробелов, сколько их требуются, чтобы переместить каретку в следующую позицию табуляции. Позиции табуляции принимаются такими, чтобы быть в каждом восьмом знакоместе.

LISTBOX (*ОКНО СО СПИСКОМ*). Обозначает список строк символов. Этот элемент управления определяется всякий раз, когда прикладная программа должна представить список наименований, типа имен файлов, из которых пользователь может выбирать. Пользователь может выбрать строку, щелкая мышью по ней. Выбранная строка выделяется, а уведомительное сообщение передается в родительское окно. Чтобы листать списки, которые являются слишком длинными для элемента управления окна, используются вертикальная или горизонтальная линейки прокрутки окна со списком. Окно со списком автоматически скрывает или показывает линейку прокрутки, по мере необходимости.

MDICLIENT. Обозначает рабочее окно *МНОГОДОКУМЕНТНОГО ИНТЕРФЕЙСА* (*MDI*). Это окно принимает сообщения, которые управляют дочерними окнами приложений *МНОГОДОКУМЕНТНОГО ИНТЕРФЕЙСА*. Рекомендованный стиль содержит — *WS_CLIPCHILDREN* и *WS_CHILD*. Чтобы создать рабочее окно *MDI*, которое позволяет пользователю при появлении листать дочерние окна MDI, нужно определить стили *WS_HSCROLL* и *WS_VSCROLL*.

SCROLLBAR (*ЛИНЕЙКА ПРОКРУТКИ*). Обозначает прямоугольник, который содержит бегунок и имеет стрелки направленные в оба конца. Линейка прокрутки посылает уведомительное сообщение своему родительскому окну всякий раз, когда пользователь щелкает мышью по элементу управления. В случае необходимости, родительское окно ответственно за модификацию позиции бегунка. Элементы управления линейкой прокрутки имеют тот же самый вид и пользуются функциями, что и линейки прокрутки, используемые в обычных окнах. Однако, в отличие от линеек прокрутки окна, линейки прокрутки элемента управления могут быть установлены для использования где-нибудь в окне, при прокрутке вводимой информации необходимой для окна. Класс линейки прокрутки включает также и элементы управления размером окна. Окно с изменяемыми размерами - это маленький прямоугольник, который пользователь может растягивать, чтобы изменить размер окна.

STATIC (*СТАТИЧЕСКИЙ ТЕКСТ*). Обозначает простое текстовое поле, окно или прямоугольник, используемый для надписей, окно или другие отдельные элементы управления. Статические элементы управления не берут никакой вводимой информации и не обеспечивают никакой выводимой информации.

Win32 API обеспечивает общие стили окна и определенные классом стили окна. Общие стили окон представляются константами, которые начинаются с префикса WS_; они могут быть объединены оператором | (ИЛИ), чтобы формировать различные типы окон, включая главные окна, диалоговые окна и дочерние окна. Определенные классом стили окна определяют вид и поведение окон, принадлежащих к предопределенным классам элемента управления, таких как окна редактирования и окна списков.

Ниже перечислены стили кнопок (в классе *BUTTON),* которые могут быть определены в параметре *dwStyle*:

- *BS_3STATE* — Создает кнопку, которая является такой же, как окошко для флажка, за исключением того, что поле окна может стать недоступным так же, как это делается при установке флажка («галочки») проверки (*checked*) или при отмене его. Используйте недоступное состояние, чтобы показать, что состояние окошка для флажка не определено.
- *BS_AUTO3STATE* — Создает кнопку, которая является таким же переключателем с тремя состояниями, за исключением того, что поле окна изменяет свое состояние, когда пользователь выбирает его. Состояние циклически проходит фазы установки флажка проверки, недоступности и отмены установки.
- *BS_AUTOCHECKBOX* — Создает кнопку, которая также является окошком для флажка, за исключением того, что состояние установки флажка проверки автоматически переключается между установленным и не установленным параметром, каждый раз, когда пользователь выбирает эту кнопку.
- *BS_AUTORADIOBUTTON* — Создает кнопку, которая то же, что и радио-кнопка, за исключением того, что, когда пользователь выбирает её, Windows автоматически устанавливает состояние кнопки в режим контроля флажка, отметив ее «галочкой» и автоматически устанавливает проверку состояния для всех других кнопок в той же самой группе без проверки флажка.
- *BS_CHECKBOX* — Создает маленькое, пустое окошко для флажка с текстом. По умолчанию, текст отображается справа от окошка. Чтобы отображать текст слева от окошка, объедините этот флажок со стилем *BS_LEFTTEXT* (или с эквивалентным стилем *BS_RIGHTBUTTON*).
- *BS_DEFPUSHBUTTON* — Создает командную кнопку, которая ведет себя подобно кнопке стиля *BS_PUSHBUTTON* и к тому же имеет жирную черную рамку. Если кнопка находится в диалоговом окне, пользователь может выбрать кнопку, нажав клавишу *ENTER,* даже тогда, когда кнопка не имеет фокуса ввода. Этот стиль полезен для предоставления пользователю возможности быстро

выбрать наиболее подходящую (заданную по умолчанию) опцию.

- *BS_GROUPBOX* — Создает прямоугольник, в котором могут быть сгруппированы другие элементы управления. Любой текст, связанный с этим стилем отображается в верхнем левом угле прямоугольника.
- *BS_LEFTTEXT* — Помещает текст слева от радио-кнопки или окошечка-переключателя, когда объединен со стилем переключателя или радио-кнопкой. Тот же самое, что и стиль *BS_RIGHTBUTTON*.
- *BS_OWNERDRAW* — Создает кнопку представляемую владельцем. Окно владельца принимает сообщение *WM_MEASUREITEM*, когда кнопка создана и сообщение *WM_DRAWITEM*, когда внешний вид кнопки изменился. Не объединяйте стиль *BS_OWNERDRAW* с любыми другими стилями кнопки.
- *BS_PUSHBUTTON* — Создает командную кнопку, которая отправляет сообщение *WM_COMMAND* окну владельца, когда пользователь выбирает эту кнопку.
- *BS_RADIOBUTTON* - Создает маленький кружок с текстом. По умолчанию, текст отображается справа от кружка. Чтобы отображать текст слева от кружка, объедините этот флажок со стилем *BS_LEFTTEXT* (или его эквивалентом — стилем *BS_RIGHTBUTTON*). Используйте радио-кнопки для групп связанного, но взаимоисключающего выбора.
- *BS_USERBUTTON* — Устаревшая, но предусматривающая совместимость с 16-разрядными версиями Windows. Базирующиеся на Win32 приложения должны использовать *BS_OWNERDRAW* взамен этого параметра.
- *BS_BITMAP* — Определяет, что кнопка отображает точечный рисунок.
- *BS_BOTTOM* — Помещает текст внизу прямоугольника кнопки.
- *BS_CENTER* — Выравнивает текст горизонтально по центру в прямоугольнике кнопки.
- *BS_ICON* — Определяет, что кнопка отображается как значок.
- *BS_LEFT* — Выравнивает слева текст в прямоугольнике кнопки. Однако, если кнопка — окошечко-переключатель или радио-кнопка, которые не имеет стиля *BS_RIGHTBUTTON*, текст выровнен вправо от переключателя или радио-кнопки.
- *BS_MULTILINE* — Переносит по словам текст кнопки в дополнительные строки, если текстовая строка слишком длинна, чтобы поместиться в одной строке в прямоугольнике кнопки.
- *BS_NOTIFY* — Дает возможность кнопке послать уведомительные сообщения *BN_DBLCLK*, *BN_KILLFOCUS* и *BN_SETFOCUS* в её родительское окно. Кнопки посылают сообщение *BN_CLICKED* независимо от того, имеет ли она этот стиль.
- *BS_PUSHLIKE* — Создает кнопку (типа переключателя, переключателя с тремя состояниями или радио-кнопки) имеющую вид и действующую подобно командной кнопке. Выпуклый вид кнопки, когда она не нажата или не выбрана, и притопленый, когда она нажата или выбрана.
- *BS_RIGHT* — Выровненный справа текст в прямоугольнике кнопки. Однако, если

кнопка — окошко для флажка или радио-кнопка, которая не имеет стиля *BS_RIGHTBUTTON*, текст выровнен по правому краю справа от окошка для флажка или радио-кнопки.

- *BS_RIGHTBUTTON* — Устанавливает кружок радио-кнопки или квадрат окошка для флажка справа от прямоугольника кнопки. Тот же самый стиль, что и *BS_LEFTTEXT*.
- *BS_TEXT* — Определяет, что кнопка отображает текст.
- *BS_TOP* — Размешает текст вверху прямоугольника кнопки.
- *BS_VCENTER* — Размещает текст в середине (вертикально) прямоугольника кнопки.

Ниже перечислены стили комбинированного окна (в классе *COMBOBOX*), которые могут быть определены в параметре *dwStyle*:

- *CBS_AUTOHSCROLL* — Автоматически прокручивает текст в поле редактирования текста вправо, когда пользователь вводит с клавиатуры символ в конце строки. Если этот стиль не установлен, принимается только текст, который помещается внутри прямоугольной границы поля.
- *CBS_DISABLENOSCROLL* — В окне со списком показывает вертикальную линейку прокрутки заблокированной, когда поле окна содержит не достаточно элементов для прокрутки. Без этого стиля, линейка прокрутки скрыта, если окно со списком содержит не достаточно элементов.
- *CBS_DROPDOWN* — Подобен *CBS_SIMPLE*, за исключением того, что окно со списком не отображается, пока пользователь не выберет значок рядом с полем редактирования текста.
- *CBS_DROPDOWNLIST* — Подобен *CBS_DROPDOWN*, за исключением того, что поле редактирования текста заменено статическим текстовым элементом, который отображает текущий выбор в окне со списком.
- *CBS_HASSTRINGS* — Определяет, что представляемое владельцем комбинированное окно содержит элементы, состоящие из строк. Комбинированное окно поддерживает память и адрес для строк, так что прикладная программа может использовать сообщение *CB_GETLBTEXT*, чтобы восстановить текст для отдельного элемента.
- *CBS_LOWERCASE* — Преобразовывает в нижний регистр любые символы верхнего регистра, введенные в поле редактирования текста комбинированного окна.
- *CBS_NOINTEGRALHEIGHT* — Определяет, что размер комбинированного окна - это точный размер, определенный прикладной программой, когда она создала комбинированное окно. Обычно, операционная система устанавливает размеры комбинированного окна таким образом, чтобы оно не отображало элементы частично.

- *CBS_OEMCONVERT* - Преобразует текст, введенный в поле редактирования текста комбинированного окна. Текст преобразуется из набора символов Windows в набор символов OEM, а затем обратно в набор Windows. Это гарантирует соответствующее символьное преобразование, когда прикладная программа вызывает функцию *CharToOem*, чтобы преобразовать строку Windows в комбинированном окне в символы OEM. Этот стиль наиболее полезен для комбинированных окон, которые содержат имена файлов и применяются только в комбинированных окнах, созданных со стилем *CBS_SIMPLE* или *CBS_DROPDOWN*.
- *CBS_OWNERDRAWFIXED* — Определяет, что владелец окна со списком ответственен за прорисовку его содержания и что элементы в окне со списком все равной высоты. Окно владельца принимает сообщение *WM_MEASUREITEM*, когда комбинированное окно создано, а сообщение *WM_DRAWITEM*, когда внешний вид комбинированного окна изменился.
- *CBS_OWNERDRAWVARIABLE* — Определяет, что владелец окна со списком ответственен за прорисовку его содержания и что элементы в окне со списком являются переменными по высоте. Окно владельца принимает сообщение *WM_MEASUREITEM* для каждого элемента комбинированного окна, когда создается комбинированное окно; окно владельца принимает сообщение *WM_DRAWITEM* тогда, когда изменился внешний вид комбинированного окна.
- *CBS_SIMPLE* — Всегда отображать окно со списком. Текущий выбор в окне со списком отображается в поле редактирования текста.
- *CBS_SORT* — Автоматически сортирует строки, введенные в окно со списком.
- *CBS_UPPERCASE* — Преобразовывает любые символы нижнего регистра в символы верхнего регистра, введенные в поле редактирования текста комбинированного окна.

Ниже перечисленные стили поля редактирования текста (в классе *EDIT)* могут быть определены в параметре *dwStyle:*

- *ES_AUTOHSCROLL* — Автоматически прокручивает текст вправо на 10 символов, когда пользователь напечатает символ в конце строчки. Когда пользователь нажимает клавишу *ENTER*, управление прокручивает весь текст обратно, чтобы установить нуль.
- *ES_AUTOVSCROLL* — Автоматический вертикальный скроллинг (перемещает текст вверх на одну страницу), когда пользователь нажимает клавишу *ENTER* на последней строчке.
- *ES_CENTER* — Выравнивает по центру текст в многостроковом поле редактирования текста.
- *ES_LEFT* — Выравнивание текста слева.
- *ES_LOWERCASE* — Преобразовывает все символы в нижний регистр, поскольку

они печатаются внутри поля редактирования текста.

- *ES_MULTILINE* — Обозначает многострочное окно редактирования текста. Значение по умолчанию — однострочное окно редактирования текста. Когда многострочное поле редактирования находится в диалоговом окне, заданная по умолчанию ответная реакция на нажим клавиши *ENTER* должна активизировать кнопку по умолчанию. Чтобы использовать клавишу *ENTER* для перевода строки, стиль используйте *ES_WANTRETURN*. Когда многострочное окно редактирования не в диалоговом окне и определен стиль *ES_AUTOVSCROLL*, поле редактирования показывает столько строчек, сколько это возможно и прокручивает вертикально, когда пользователь нажимает клавишу *ENTER*. Если вы не определяете *ES_AUTOVSCROLL*, окно редактирования показывает столько строчек, сколько это возможно и подает звуковой сигнал, если пользователь нажимает клавишу *ENTER*, но больше ни строчки не может отобразиться в окне. Если вы определяете стиль *ES_AUTOHSCROLL*, многострочное окно редактирования автоматически горизонтально прокручивается, когда каретка проходит за правый край элемента управления. Чтобы запустить новую строку, пользователь должен нажать клавишу *ENTER*. Если вы не определяете *ES_AUTOHSCROLL*, элемент управления, когда это необходимо, автоматически переносит без разрыва слова в начало следующей строки. Новая строка образуется и тогда, если пользователь нажимает клавишу *ENTER*. Размер окна определяет позицию перехода слова на новую строку. Если размер окна изменяется, изменяется позиция перехода на новую строку, а текст восстанавливается. Многострочное окно редактирования текста может иметь линейки прокрутки. Окно редактирования с линейками прокрутки обрабатывают свои собственные сообщения от линейки прокрутки. Окно редактирования без линеек прокрутки, прокручивают текст, как описано выше, и обрабатывают любые сообщений прокрутки, посланные родительским окном.
- *ES_NOHIDESEL* — Отрицает заданное по умолчанию поведение для поля редактирования текста. Заданное по умолчанию поведение скрывает выбор, когда элемент управления теряет фокус ввода и инвертирует выбор, когда панель управления принимает фокус ввода. Если вы определяете *ES_NOHIDESEL*, выбранный текст инвертируется, даже если панель управления не имеет фокуса.
- *ES_NUMBER* — Позволяет ввести в поле редактирования только цифры.
- *ES_OEMCONVERT* - Преобразует текст, введенный в окно редактирования. Текст преобразуется из набора символов Windows — в набор символов OEM, а затем обратно — в набор Windows. Это гарантирует соответствующее символьное преобразование, когда из приложения вызывается функция *CharToOem*, чтобы преобразовать строку Windows в окне редактирования в символы OEM. Этот стиль наиболее полезен для окон редактирования текста, которые содержат имена файлов.

- *ES_PASSWORD* — Отображает звездочку (*) вместо каждого символа, введенного с клавиатуры в окно редактирования. Вы можете использовать сообщение *EM_SETPASSWORDCHAR*, чтобы заменить ею символ, который отображается.
- *ES_READONLY* — Не допускает пользователя к вводу или редактированию текста в окне редактирования.
- *ES_RIGHT* — Выравнивает по правому краю текст в многострочном окне редактирования.
- *ES_UPPERCASE* — Преобразует все символы в символы верхнего регистра, когда они вводятся в окно редактирования.
- *ES_WANTRETURN* — Определяет, чтобы служебный код возврата каретки был вставлен тогда, когда пользователь нажимает клавишу *ENTER* при вводе текста в многострочное поле редактирования текста в диалоговом окне. Если вы не определяете этот стиль, нажимая клавишу *ENTER*, вы получите тот же самый эффект, словно нажали заданную по умолчанию командную кнопку диалогового окна. Этот стиль не имеет никакого влияния в однострочном окне редактирования.

Следующие стили элемента управления окна со списком (в классе *LISTBOX)* могут быть определены в параметре *dwStyle:*

- *LBS_DISABLENOSCROLL* — Показывает заблокированную вертикальную линейку прокрутки в окне со списком, когда поле окна не содержит достаточно элементов для прокрутки. Если вы не определяете этот стиль, линейка прокрутки скрыта, когда окно со списком не содержит достаточно элементов.
- *LBS_EXTENDEDSEL* — Позволяет многочисленным элементам быть выбранными, при помощи использования клавиши *SHIFT* и мыши или специальной комбинации клавиш.
- *LBS_HASSTRINGS* — Определяет, что окно со списком содержит элементы, состоящие из строк. Окно со списком сохраняет память и адреса строк, так что прикладная программа может использовать сообщение *LB_GETTEXT*, чтобы восстановить текст для отдельного элемента. По умолчанию, все окна со списком за исключением окон со списком предоставленных владельцем имеют этот стиль. Вы можете создать предоставляемое владельцем окно со списком с ним или без этого стиля.
- *LBS_MULTICOLUMN* — Определяет многостолбцовое окно со списком, которое прокручивается горизонтально. Сообщение *LB_SETCOLUMNWIDTH* устанавливает ширину столбцов.
- *LBS_MULTIPLESEL* — Включает или выключает выбор последовательности символов каждый раз, когда пользователь одним или двойным щелчком мыши активизирует строку символов в окне со списком. Пользователь может выбрать любое число строк.

- *LBS_NODATA* — Определяет «отсутствие данных» в окне со списком. Этот стиль определяется тогда, когда число элементов в окне со списком может превысить одну тысячу. Окно со списком с «отсутствующими данными» должно иметь также стиль *LBS_OWNERDRAWFIXED*, но не должно иметь стилей *LBS_SORT* или *LBS_HASSTRINGS*. Окно со списком с «отсутствующими данными» имеет сходство с окном со списком предоставляемым владельцем за исключением того, что оно не содержит ни строковых или растровых данных для элемента. Команды «добавить», «вставить» или «удалить» элемент всегда игнорируют любые передаваемые элементы данных; запрос на поиск строки внутри окна всегда терпит неудачу. Windows посылает сообщение *WM_DRAWITEM* окну владельцу, когда элемент должен быть прорисован. Элемент ID *(itemID)* — член структуры *DRAWITEMSTRUCT*, переданный с сообщением *WM_DRAWITEM*, определяет номер строки элемента, который будет прорисован. Окно списка с «отсутствующими данными» не посылает сообщение *WM_DELETEITEM*.
- *LBS_NOINTEGRALHEIGHT* — Определяет, что размер окна со списком соответствует размеру, определенному прикладной программой, когда она создавала окно со списком. Обычно, Windows устанавливает величину окна со списком так, чтобы оно не отображало элементы частично.
- *LBS_NOREDRAW* — Определяет, что вид окна со списком не модифицируется, когда производятся изменения. Вы можете в любое время изменить этот стиль, посылая сообщение *WM_SETREDRAW*.
- *LBS_NOSEL* — Определяет, что окно со списком содержит элементы, которые могут просматриваться, но не выбираться.
- *LBS_NOTIFY* — Сообщает родительскому окну о входящем сообщении всякий раз, когда пользователь щелкает мышью или дважды щелкает по строке в окне списка.
- *LBS_OWNERDRAWFIXED* — Определяет, что владелец окна со списком ответственен за прорисовку его содержания и что элементы в окне со списком появляются одинаковой высоты. Окно владельца принимает сообщение *WM_MEASUREITEM*, когда окно со списком создано, а сообщение *WM_DRAWITEM*, когда внешний вид окна изменился.
- *LBS_OWNERDRAWVARIABLE* — Определяет, что владелец окна со списком ответственен за прорисовку его содержания и что элементы в окне со списком появляются переменными по высоте. Окно владельца принимает сообщение *WM_MEASUREITEM* для каждого элемента в окне со списком, когда оно создано, а сообщение *WM_DRAWITEM*, когда внешний вид окна изменился.
- *LBS_SORT* — Сортирует строки в окне со списком по алфавиту.
- *LBS_STANDARD* — Сортирует строки в окне со списком в алфавитном порядке. Родительское окно принимает входящее сообщение всякий раз, когда пользователь щелкает мышью или дважды щелкает по строке. Окно со списком

имеет, рамку со всех сторон.

- *LBS_USETABSTOPS* — Дает возможность окну списка распознавать и развернуть символы в виде таблицы при прорисовке его строк. По умолчанию таблица занимает 32 единицы измерения диалогового окна. Единица измерения диалогового окна - горизонтальное или вертикальное расстояние. Одна горизонтальная единица диалогового окна равна четвертой части текущей единицы измерения габаритов диалогового окна. Windows вычисляет эти единицы измерения, основанные на высоте и ширине шрифта существующей системы. Функция *GetDialogBaseUnits* возвращает значение текущей базовой единицы измерения диалогового окна в пикселах.
- *LBS_WANTKEYBOARDINPUT* — Определяет, что владелец окна списка принимает сообщения *WM_VKEYTOITEM* всякий раз, когда пользователь нажимает клавишу, а окно со списком имеет фокус ввода. Это дает возможность прикладной программе выполнить специальную обработку при вводе с клавиатуры.

Следующие стили линейки прокрутки (в классе *SCROLLBAR*) могут быть определены в параметре *dwStyle*:

- *SBS_BOTTOMALIGN* — Выравнивает нижнюю кромку линейки прокрутки с нижней кромкой прямоугольника, определенного параметрами *x*, *y*, *nWidth* и *nHeight*. Линейка прокрутки имеет заданную по умолчанию высоту для системных линеек прокрутки. Используйте этот стиль со стилем *SBS_HORZ*.
- *SBS_HORZ* — Обозначает горизонтальную линейку прокрутки. Если стили ни *SBS_BOTTOMALIGN*, ни *SBS_TOPALIGN* не определены, линейка прокрутки имеет высоту, ширину и позицию, определенные *x*, *y*, *nWidth* и *nHeight*.
- *SBS_LEFTALIGN* — Выравнивает левый край линейки прокрутки с левым краем прямоугольника, определенного параметрами *x*, *y*, *nWidth* и *nHeight*. Линейка прокрутки имеет заданную по умолчанию ширину для системных линеек прокрутки. Используйте этот стиль с *SBS_VERT* стилем.
- *SBS_RIGHTALIGN* — Выравнивает правый край линейки прокрутки с правым краем прямоугольника, определенного параметрами *x*, *y*, *nWidth* и *nHeight*. Линейка прокрутки имеет заданную по умолчанию ширину для системных линеек прокрутки. Используйте этот стиль с *SBS_VERT* стилем.
- *SBS_SIZEBOX* — Обозначает размер окна. Если вы не определяете ни *SBS_SIZEBOXBOTTOMRIGHTALIGN*, ни *SBS_SIZEBOXTOPLEFTALIGN* стиль, размер окна имеет высоту, ширину и позицию, определенную параметрами *x*, *y*, *nWidth* и *nHeight*.
- *SBS_SIZEBOXBOTTOMRIGHTALIGN* — Выравнивает размер нижнего правого угла окна с нижним правым углом прямоугольника, определенного параметрами *x*, *y*, *nWidth* и *nHeight*. Размер окна имеет заданный по умолчанию размер для

системы размера окон. Используйте этот стиль с *SBS_SIZEBOX* стилем.

- *SBS_SIZEBOXTOPLEFTALIGN* - Выравнивает размер верхнего левого угла окна с левым верхним углом прямоугольника, определенного параметрами *x*, *y*, *nWidth* и *nHeight*. Размер окна имеет заданный по умолчанию размер для системы размера окон. Используйте этот стиль с *SBS_SIZEBOX* стилем.
- *SBS_SIZEGRIP* — Подобен стилю *SBS_SIZEBOX*, но с выпуклой рамкой.
- *SBS_TOPALIGN* — Выравнивает верхний край линейки прокрутки с верхним краем прямоугольника, определенного параметрами *x*, *y*, *nWidth* и *nHeight*. Линейка прокрутки имеет заданную по умолчанию высоту для системы линеек прокрутки. Используйте этот стиль с *SBS_HORZ* стилем.
- *SBS_VERT* — Обозначает вертикальную линейку прокрутки. Если вы не определяете ни *SBS_RIGHTALIGN*, ни *SBS_LEFTALIGN* стиль, линейка прокрутки имеет высоту, ширину и позицию, определенную параметрами *x*, *y*, *nWidth* и *nHeight*.

Следующие стили статического текста (в классе *STATIC*) могут быть определены в параметре *dwStyle*. Статический текст может иметь только один из этих стилей:

- *SS_BITMAP* — Определяет, что в статическом тексте должен отобразиться точечный рисунок. Текст кода ошибки — имя точечного рисунка (не имя файла) определенного в другом месте файла ресурса. Стиль игнорирует параметры *nWidth* и *nHeight*; элемент управления автоматически устанавливает собственные размеры, чтобы поместить точечный рисунок.
- *SS_BLACKFRAME* — Определяет окно с рамкой, использующей тот же самый цвет, как и у рамки основного окна. Этот цвет черный по умолчанию в системе цветов Windows.
- *SS_BLACKRECT* — Определяет прямоугольник, заполненный текущим цветом рамки окна. По умолчанию этот цвет черный в системе цветов Windows.
- *SS_CENTER* — Определяет простой прямоугольник и выравнивает по центру текст кода ошибки в прямоугольнике. Текст форматируется перед отображением его на экране. Слова, которые выходят за пределы конца строки автоматически переносятся в начало следующей центрированной строки.
- *SS_CENTERIMAGE* — Определяет, что средняя точка статического текста со стилем *SS_BITMAP* или *SS_ICON* должна остаться фиксированной, когда элемент управления изменяется. Четыре стороны корректируются так, чтобы поместить новый точечный рисунок или пиктограмму. Если статический текст имеет стиль *SS_BITMAP*, а точечный рисунок меньше чем рабочая область элемента управления, рабочая область заполняется цветом пикселя левого верхнего угла точечного рисунка. Если статический текст имеет стиль *SS_ICON*, пиктограмма появляется, но не окрашивает рабочую область.

- *SS_GRAYFRAME* — Определяет поле окна с рамкой, выведенной тем же самым цветом, что и экранный фон (рабочий стол). По умолчанию в системе цветов Windows этот цвет серый.
- *SS_GRAYRECT* — Определяет прямоугольник, заполненный текущим экранным цветом фона. По умолчанию в системе цветов Windows этот цвет серый.
- *SS_ICON* — Определяет пиктограмму, отображаемую в диалоговом окне. Данный текст - имя пиктограммы (не имя файла) определенный в другом месте файла ресурса. Стиль игнорирует параметры *nWidth* и *nHeight*; пиктограмма автоматически устанавливает свою величину.
- *SS_LEFT* — Определяет простой прямоугольник и выравнивание по левому краю текста, помещенного в прямоугольнике. Текст форматируется перед его отображением. Слова, которые выходят за пределы конца строки автоматически переносятся в начало следующей выровненной по левой границе строки.
- *SS_LEFTNOWORDWRAP* — Определяет простой прямоугольник и выравнивание по левому краю текста, помещенного в прямоугольнике. Планшеты расширяются, но слова не переносятся. Текст, который выходит за пределы конца строки, отсекается.
- *SS_METAPICT* — Определяет, что изображение метафайла должно отобразиться в статическом тексте. Данный текст — имя изображения (не имя файла) определенный в другом месте в файле ресурса. Статический текст метафайла имеет фиксированный размер; изображение метафайла масштабируется, чтобы приспособить рабочую область статического текста.
- *SS_NOPREFIX* — Предотвращает интерпретацию любого символа амперсанда (&) в тексте элемента управления как символа префикса акселератора. Они отображаются с удаленным амперсандом и следующим за ним подчеркнутым символом в строке. Этот стиль статического текста может быть включен с любым из определенных статических текстов. Приложение может объединять *SS_NOPREFIX* с другими стилями, используя побитовый оператор OR = ИЛИ (|). Это может быть полезно, когда имена файлов или другие строки, которые могут содержать амперсанд (&) должны отображаться в статическом элементе управления диалогового окна.
- *SS_NOTIFY* — Посылает родительскому окну уведомительные сообщения *STN_CLICKED* и *STN_DBLCLK*, когда пользователь щелкает или дважды щелкает мышью по элементу управления.
- *SS_RIGHT* - Определяет простой прямоугольник и выравнивает по правому краю текста помещенный в прямоугольнике. Текст форматируется перед его отображением на экране. Слова, которые выходят за пределы конца строки переносятся в начало следующей строки выровненной по правой границе.
- *SS_RIGHTIMAGE* — Определяет, что угол правой нижней части статического текста со стилем *SS_BITMAP* или *SS_ICON* должен остаться фиксированным,

когда элемент управления изменяется. Только верхняя и левая стороны корректируются, чтобы поместить новый точечный рисунок или пиктограмму.

- *SS_SIMPLE* — Определяет простой прямоугольник и отображает одиночную строку выровненного по левой границе текста в прямоугольнике. Текстовая строка не может быть, сокращена или изменена в любом случае. Родительское окно панели управления или диалоговое окно не должны обрабатывать сообщение *WM_CTLCOLORSTATIC*.
- *SS_WHITEFRAME* — Определяет поле окна с рамкой, выведенной тем же самым цветом как фон окна. По умолчанию, в системе цветов Windows - этот цвет белый.
- *SS_WHITERECT* — Определяет прямоугольник, заполненный текущим цветом фона окна. По умолчанию, в системе цветов Windows - этот цвет белый.

Ниже перечислены стили диалогового окна, которые могут быть определены в параметре *dwStyle*:

- *DS_3DLOOK* — Обеспечивает диалоговое окно не полужирным шрифтом и выводит трехмерные рамки вокруг элементов управления окна в блоке диалога.
- *DS_3DLOOK* — Этот стиль требуется только базирующимся на Win32 приложениям компилируемым для версий Windows ранее, чем Windows 95 или Windows NT 4.0. Система автоматически применяет трехмерный вид к диалоговым окнам, созданным приложениями, компилируемыми для текущих версий Windows.
- *DS_ABSALIGN* — Указывает, что координаты диалогового окна — экранные координаты; иначе, операционная система Windows принимает их за координаты пользователя.
- *DS_CENTER* — Выравнивает по центру диалоговое окно в рабочей области; то есть в области, не загораживаемой панелью.
- *DS_CENTERMOUSE* — Выравнивает по центру курсор мыши в диалоговом окне.
- *DS_CONTEXTHELP* — Включает вопросительный знак в строке заголовка диалогового окна. Когда пользователь щелкает мышью по вопросительному знаку, курсор изменятся на вопросительный знак со стрелкой-указателем. Если пользователь затем щелкает мышью по элементу управления в диалоговом окне, элемент управления принимает сообщение *WM_HELP*. Элемент управления должен передать сообщение для диалоговой процедуры, которая должна вызвать функцию WinHelp, использующую команду *HELP_WM_HELP*. Приложение «Справка» *(Help)* отображает всплывающее окно, которое обычно содержит справку об элементе управления. Обратите внимание, что *DS_CONTEXTHELP* - только метка-заполнитель. Когда диалоговое окно создано, система проверяет наличие *DS_CONTEXTHELP* и, если она имеется, добавляет *WS_EX_CONTEXTHELP* к расширенному стилю диалогового окна.

WS_EX_CONTEXTHELP не может использоваться со стилями *WS_MAXIMIZEBOX* или *WS_MINIMIZEBOX*.

- *DS_CONTROL* — Создает диалоговое окно, которое работает также как дочернее окно другого диалогового окна, очень похожее на страницу в окне свойств. Этот стиль позволяет пользователю перемещаться среди элементов управления дочернего диалогового окна, использовать его клавиши-ускорители, и так далее.
- *DS_FIXEDSYS* – Использует *SYSTEM_FIXED_FONT* вместо *SYSTEM_FONT*.
- *DS_LOCALEDIT* — Применяется только для 16-разрядных прикладных программ. Этот стиль управляет элементами редактирования в диалоговом окне, чтобы зарезервировать память в сегменте данных прикладной программы. Иначе, элементы редактирования резервируют память объекта глобальной памяти.
- *DS_MODALFRAME* — Создает диалоговое окно с модальной рамкой диалогового окна, которая может быть объединена со строкой заголовка и меню окна, путем определения стилей *WS_CAPTION* и *WS_SYSMENU*.
- *DS_NOFAILCREATE* — Создает диалоговое окно, даже если происходят ошибки - например, если дочернее окно не может быть создано или если система не может создать специальный сегмент данных для элементов редактирования.
- *DS_NOIDLEMSG* — Подавляет сообщения *WM_ENTERIDLE*, которое Windows иначе послала бы владельцу диалогового окна, в то время как диалоговое окно отображается на экране.
- *DS_SETFONT* — Указывает, что шаблон диалогового окна (структура *DLGTEMPLATE*) содержит два дополнительных элемента, определяющих имя шрифта и размер в пунктах. Соответствующий шрифт используется, чтобы отображать текст внутри рабочей области диалогового окна и внутри элементов управления диалогового окна. Windows передает дескриптор шрифта диалоговому окну и каждому элементу управления, посылая им сообщение *WM_SETFONT*.
- *DS_SETFOREGROUND* — Не применяется в 16-разрядных версиях Microsoft Windows. Этот стиль приводит диалоговое окно в активный режим. Внутри Windows вызывает для диалогового окна функцию *SetForegroundWindow*.
- *DS_SYSMODAL* — Создает системно-модальное диалоговое окно. Этот стиль заставляет диалоговое окно иметь стиль *WS_EX_TOPMOST*, но в остальном, он не имеет никакого влияния на диалоговое окно или поведение других окон в системе, когда отображается диалоговое окно.

Функция *ShowWindow* устанавливает режим отображения окна:

```
#include <windows.h>

// Функция установки режима отображения окна
//
// FUNCTION: BOOL ShowWindow(HWND, int)
```

```
//
// PARAMETERS: [in]  hWindow       – указатель на окно
//             [in]  nCmdShow      – режим отображения
//
// RETURN VALUE: ненулевое значение  - если окно было видимым
//               0                   - если окно было скрыто
//
// COMMENTS:  Значение параметра nCmdShow, равное SW_HIDE указывает на скрытие окна,
//            SW_SHOW — на активизацию окна, SW_MINIMIZE – на минимизацию окна,
//            SW_MAXIMIZE – на развертывание окна, SW_RESTORE – на показ окна с его
//                                                 оригинальными параметрами
//
BOOL ShowWindow(HWND hWindow, int nCmdShow);
```

Функция *UpdateWindow* обновляет указанное окно, посылая ему сообщение *WM_PAINT*. Это сообщение посылается непосредственно процедуре указанного окна, обходя очередь других сообщений.

```
#include <windows.h>

// Функция обновления содержимого окна
//
// FUNCTION: BOOL UpdateWindow(HWND)
//
// PARAMETERS: [in]  hWindow       – указатель на окно
//
// RETURN VALUE: ненулевое значение  - в случае успеха
//               0                   - в случае неудачи
//
// COMMENTS:  Для получения кода ошибки нужно вызвать GetLastError
//
BOOL UpdateWindow(HWND Wnd);
```

Функция *DestroyWindow* уничтожает определенное окно. Функция посылает сообщения WM_DESTROY и WM_NCDESTROY окну, чтобы дезактивировать его и удалить фокус клавиатуры. Функция также уничтожает меню окна, очищает очередь потоков сообщений, уничтожает таймеры, удаляет монопольное использование буфера обмена и разрывает цепочку просмотра окон буфера обмена (если окно имеет наверху цепочку просмотров). Если определенное окно – родитель или владелец окон, *DestroyWindow* автоматически уничтожает связанные дочерние или находящиеся в собственности окна, когда она уничтожает окно владельца или родителя. Функция сначала уничтожает дочерние или находящиеся в собственности окна, и затем она уничтожает окно владельца или родителя.

```
#include <windows.h>

// Функция уничтожения окна
//
// FUNCTION: BOOL DestroyWindow(HWND)
//
// PARAMETERS: [in]  hWindow       – хэндл уничтожаемого окна
//
```

```
// RETURN VALUE: ненулевое значение  - в случае успеха
//               0                   - в случае неудачи
//
// COMMENTS:  Для получения кода ошибки нужно вызвать GetLastError
//            Текущий поток не может уничтожить окно, созданное другим потоком
//
BOOL DestroyWindow(HWND hWnd);
```

Описание функций WinAPI по управлению обменом сообщениями ОС Windows

Программы в ОС Windows управляются сообщениями. Все действия пользователей перехватываются системой и преобразуются в сообщения, направляемые программе, которая владеет окнами, к которым обращены действия пользователя. У каждой программы, состоящей хотя бы из одного потока, есть несколько собственных очередей сообщений: очередь асинхронных сообщений, очередь синхронных сообщений, очередь ответных сообщений и очередь виртуального ввода, куда и направляются все сообщения, касающиеся действий с окнами. В каждой программе есть главный цикл, который состоит из получения следующего сообщения и его обработки с помощью внутренней процедуры, соответствующей данному типу сообщений. Иногда некоторые сообщения вызывает эти процедуры, минуя очереди сообщений.

Асинхронные сообщения

С набором сообщений работает несколько функций Win32 API. Сообщения ставятся в очередь асинхронных сообщений при помощи функции

```
#include <windows.h>
// Функция асинхронной отправки сообщения
//
// FUNCTION: BOOL PostMessage(HWND, UINT, WPARAM, LPARAM)
//
// PARAMETERS:  [in]  hWindow    - хэндл окна-получателя сообщения
//              [in]  Message    - передаваемое сообщение
//              [in]  wParameter – дополнительная информация для сообщения
//              [in]  lParameter – дополнительная информация для сообщения
//
// RETURN VALUE: ненулевое значение - в случае успеха
//               0                  - в случае неудачи
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//                нужно вызвать GetLastError
//            Для Windows 2000/XP длина очереди ограничена 10 тысячами сообщений
//
BOOL PostMessage(HWND hWindow, UINT Message, WPARAM wParameter, LPARAM lParameter);
```

При вызове этой функции определяется поток, создавший окно с дескриптором *hWindow*. Далее выделяется память для параметров сообщения и производится добавление в очередь асинхронных сообщений. Возврат из этой функции происходит немедленно, и вероятно, что окно даже не получит данное

сообщение. Кстати, упоминавшаяся ранее функция *PostQuitMessage* тоже добавляет сообщение в очередь асинхронных сообщений потока. Она определена так:

```
#include <windows.h>
// Функция асинхронной отправки потоку сообщения о завершении приложения
//
// FUNCTION: VOID PostQuitMessage(int)
//
// PARAMETERS:  [in]  nExitCode  - код завершения приложения
//
// RETURN VALUE: функция не возвращает значения
//
// COMMENTS:  Значение параметра nExitCode используется как wParameter
//            сообщения WM_QUIT
//            Обычно используется для ответа на сообщение WM_DESTROY
//
VOID PostQuitMessage(int);
```

Сообщение можно поставить в очередь асинхронных сообщений потока и вызовом *PostThreadMessage*:

```
#include <windows.h>
// Функция асинхронной отправки потоку сообщения
//
// FUNCTION: BOOL PostThreadMessage(DWORD, UINT, WPARAM, LPARAM)
//
// PARAMETERS:  [in]  dwThreadId – идентификатор потока,
//                                 которому посылается сообщение
//              [in]  Message    - передаваемое сообщение
//              [in]  wParameter – дополнительная информация для сообщения
//              [in]  lParameter – дополнительная информация для сообщения
//
// RETURN VALUE: ненулевое значение - в случае успеха
//               0                  - в случае неудачи
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//                нужно вызвать GetLastError
//            Для Windows 2000/XP длина очереди ограничена 10 тысячами сообщений
//
BOOL PostThreadMessage(DWORD dwThreadId, UINT Message
                       , WPARAM wParameter, LPARAM lParameter
                      );
```

Синхронные сообщения

Оконное сообщение можно отправить любой оконной процедуре вызовом функции *SendMessage*, которая производит добавление в очередь синхронных сообщений.

```
#include <windows.h>
// Функция синхронной отправки сообщения
//
// FUNCTION: LRESULT SendMessage(HWND, UINT, WPARAM, LPARAM)
//
// PARAMETERS:  [in]  hWindow    - хэндл окна-получателя сообщения
```

```
//              [in]  Message    - передаваемое сообщение
//              [in]  wParameter - дополнительная информация для сообщения
//              [in]  lParameter - дополнительная информация для сообщения
//
// RETURN VALUE: Возвращаемое значение зависит от отправляемого сообщения
//
// COMMENTS:  нет
//
LRESULT SendMessage(HWND hWindow, UINT Message
                    , WPARAM wParameter, LPARAM lParameter
                   );
```

Оконная процедура обработает сообщение *Message*, и только после этого вернет управление. Пока сообщение не обработано, поток находится в состоянии ожидания. Поток, обрабатывающий синхронное сообщение, может содержать бесконечный цикл, тогда поток-отправитель «со спокойной совестью» может «зависнуть». Избежать таких ситуаций можно при помощи нескольких функций WinAPI. Одна из них ожидает в течение некоторого времени ответа от другого потока.

```
#include <windows.h>
// Функция синхронной отправки сообщения с ожиданием ответа
//
// FUNCTION: LRESULT SendMessageTimeout(HWND, UINT
                                        , WPARAM, LPARAM
                                        , UINT, UINT, PDWORD_PTR
                                       )
//
// PARAMETERS:  [in]  hWindow    - хэндл окна-получателя сообщения
//              [in]  Message    - передаваемое сообщение
//              [in]  wParameter - дополнительная информация для сообщения
//              [in]  lParameter - дополнительная информация для сообщения
//              [in]  uFlags     - способ отправки сообщения
//              [in]  uTimeout   - время ожидания ответа в миллисекундах
//              [out] pResults   - возвращаемое значение оконной процедуры
//
// RETURN VALUE: ненулевое значение - в случае успеха
//               0                  - в случае неудачи
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//                нужно вызвать GetLastError
//            Если эта функция вернула ERROR_TIMEOUT, это значит, что
//                  SendMessageTimeout не дождалась ответа за установленное время
//
LRESULT SendMessageTimeout
(
    HWND hWindow
    , UINT Message
    , WPARAM wParameter, LPARAM lParameter
    , UINT uFlags, UINT uTimeout
    , PDWORD_PTR pResult
);
```

В параметре *uFlags* указываются флаги: *SMTO_NORMAL*, *SMTO_ABORTIFHUNG* (если поток завис, вернуть управление), *SMTO_NOTIMEOUTIFNOTHUNG* (если поток не завис, игнорировать ограничение по времени), *SMTO_SMTO_BLOCK* (не обрабатывать другие синхронные сообщения, пока функция не вернет управление). Последний флаг, однако, может «помочь» потокам перейти в состояние взаимной блокировки.

Описанная далее функция также предназначена для отправки оконных сообщений.

```
#include <windows.h>
// Функция отправки сообщения с вызовом специальной процедуры
//
// FUNCTION: BOOL SendMessageCallback(HWND, UINT
                                      , WPARAM, LPARAM
                                      , SENDASYNCPROC
                                      , ULONG_PTR
                                     )
//
// PARAMETERS:  [in]  hWindow    - хэндл окна-получателя сообщения
//              [in]  Message    - передаваемое сообщение
//              [in]  wParameter - дополнительная информация для сообщения
//              [in]  lParameter - дополнительная информация для сообщения
//              [in]  lpCallback - асинхронная процедура
//              [out] dwData     - данные для передачи оконной процедуре
//
// RETURN VALUE: ненулевое значение - в случае успеха
//               0                  - в случае неудачи
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//                нужно вызвать GetLastError
//
BOOL SendMessageCallback(HWND hWindow, UINT Message
                         , WPARAM wParameter, LPARAM lParameter
                         , SENDASYNCPROC lpCallback
                         , ULONG_PTR dwData
                        );
```

При вызове потоком этой функции сообщение добавляется в очередь синхронных сообщений потоком-приемником, а управление сразу же возвращается потоку-отправителю. По окончании обработки сообщения приемник асинхронно отправляет свое сообщение в очередь ответных сообщений. Чтобы поток-отправитель смог получить этот ответ должна быть предусмотрена функция, имеющая следующий прототип.

```
#include <windows.h>
VOID CALLBACK ResCallBack
(
    HWND hWindow,         // хэндл окна
    , UINT Message        // передаваемое сообщение
    , ULONG_PTR dwData    // данные для передачи оконной процедуре
    , LRESULT lResult     // результат обработки сообщения от оконной процедуры
);
```

Адрес этой функции передается в параметре *lpCallBack* функции *SendMessageCallback*. При вызове *ResCallBack* ей передается одноименный параметр *dwData* из *SendMessageCallback*. Параметр *lResult* содержит результат обработки сообщения от оконной процедуры.

Еще одна функция Win32 API, предназначенная для обмена сообщениями между потоками:

```
#include <windows.h>
// Функция синхронной отправки сообщения
//
// FUNCTION: BOOL SendNotifyMessage(HWND, UINT, WPARAM, LPARAM)
//
// PARAMETERS:  [in]  hWindow    - хэндл окна-получателя сообщения
//              [in]  Message    - передаваемое сообщение
//              [in]  wParameter – дополнительная информация для сообщения
//              [in]  lParameter – дополнительная информация для сообщения
//
// RETURN VALUE: ненулевое значение - в случае успеха
//               0                  - в случае неудачи
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//                нужно вызвать GetLastError
//
BOOL SendNotifyMessage(HWND hWindow, UINT Message
                       , WPARAM wParameter, LPARAM lParameter
                      );
```

Она также добавляет элемент в очередь синхронных сообщений и немедленно возвращает управление вызывающему потоку, то есть ее поведение подобно функции *PostMessage*. Однако есть и отличия. Если *SendNotifyMessage* посылает сообщение окну другого потока, то они извлекаются из очереди раньше сообщений, отправленных *PostMessage*, то есть имеет перед ними приоритет.

Если же посылается сообщение тому окну, которое создано потоком-отправителем, то управление не возвращается до окончания обработки сообщения, что очень похоже на поведение функции *SendMessage*.

С общих позиций, большинство синхронных сообщений посылается окну просто для уведомления об изменении состояния, чтобы оно каким-то образом отреагировало перед продолжением работы. Например, таковым является сообщение *WM_DESTROY*. Система не прерывает работу, чтобы процедура окна могла его обработать, тогда как сообщение *WM_CREATE* вызывает перерыв до окончания его обработки.

Ответные сообщения

Четвертая функция Win API, связанная с обработкой сообщений, достойная упоминания

```
#include <windows.h>
```

```
// Функция отправки ответных сообщения
//
// FUNCTION: BOOL ReplyMessage(LRESULT)
//
// PARAMETERS:  [in]  lResult    - результат обработки сообщения
//
// RETURN VALUE: ненулевое значение – если обрабатывалось сообщение, посланное
//                                    другим процессом или потоком
//               0                  - если не обрабатывалось сообщение, посланное
//                                    другим процессом или потоком
//
// COMMENTS:  Данная функция отвечает только на вызовы SendMessage
//
BOOL ReplyMessage(LRESULT lResult);
```

Эта функция вызывается потоком, принимающим оконное сообщение. Ответ (результат обработки) асинхронно помещается в очередь ответных сообщений потока-отправителя, а тот может теперь получить результат с помощью параметра *lResult* и продолжить свою работу. Функция возвращает *TRUE*, если обрабатывается межпоточное сообщение, а *FALSE* используется для внутрипоточных сообщений. Для того чтобы узнать является ли сообщение меж- или внутрипоточным, можно вызвать функцию

```
#include <windows.h>
// Проверка, является ли сообщение внутри- или межпоточным
//
// FUNCTION: BOOL InSendMessage(VOID)
//
// PARAMETERS:  нет
//
// RETURN VALUE: ненулевое значение – межпоточное сообщение
//               0                  - внутрипоточное сообщение
//
// COMMENTS:  нет
//
BOOL InSendMessage(VOID);
```

Возвращаемые значения: *TRUE*, если обрабатывается межпоточное синхронное сообщение, и *FALSE* при обработке синхронного и асинхронного внутрипоточного сообщения.

Некоторые из функций, предназначенных для посылки и приема сообщений, были описаны выше. Здесь перечислены некоторые сообщения, которые могут посылаться окну, в дополнение тем, что давались выше (*WM_CREATE*, *WM_DESTROY* и другие):

- *WM_MOUSEMOVE* — посылается окну, когда курсор меняет свою позицию. При этом в процедуре обработки сообщений (оконной процедуре) надо обязательно учитывать, что ее «первый» (хотя, если говорить точнее, то этот аргумент третий) параметр *FirstParametr* будет содержать флаги виртуальных клавиш, а «второй» (то есть четвертый, *SecondParameter*) — координаты

курсора. Для их получения можно воспользоваться макросом *MAKEPOINTS*.

- *WM_NCMOUSEMOVE* — посылается окну, когда курсор перемещается над неклиентской областью окна. Параметры обозначают координаты курсора и так называемые *hit-test* значения, они будут описаны ниже.
- *WM_NCHITTEST* — посылается окну, когда перемещается курсор мыши или нажимается/освобождается кнопка мыши. Параметры: координаты курсора. Возвращает *hit-test* значения.
- *WM_NCLBUTTONUP* — посылается окну, когда пользователь отжимает управляющую кнопку мыши, ее курсор находится над неклиентской областью окна. Параметры: *hit-test* значение и позиция курсора.
- *WM_NCLBUTTONDOWN* — посылается окну, когда пользователь нажимает управляющую кнопку мыши, ее курсор находится над неклиентской областью окна. Параметры: *hit-test* значение и позиция курсора.
- *WM_NCRBUTTONUP* и *WM_NCRBUTTONDOWN* аналогичны двум предыдущим, только «работают» с вспомогательной кнопкой мыши.
- *WM_NCHITTEST* посылается окну, когда перемещается курсор мыши или нажимается/отжимается одна из ее кнопок. Параметры — координаты курсора.

Hit-test значения демонстрируют, в каком месте окна находится курсор:

• *HTBORDER*	На границе окна
• *HTBOTTOM*	На нижней горизонтальной границе окна
• *HTBOTTOMLEFT*	В левом нижнем углу границы окна
• *HTBOTTOMRIGHT*	В правом нижнем углу границы окна
• *HTCAPTION*	На заголовке окна
• *HTCLIENT*	На клиентской области окна
• *HTERROR*	Фон экрана или разделительная линия между двумя окнами
• *HTGROWBOX*	В области изменения размеров окна
• *HTHSCROLL*	На горизонтальной полосе прокрутки
• *HTLEFT*	На левой границе окна
• *HTMENU*	В меню
• *HTNOWHERE*	Фон экрана или разделительная линия между двумя окнами
• *HTREDUCE*	На значке «Свернуть»
• *HTRIGHT*	На правой границе окна
• *HTSIZE*	В области изменения размеров окна
• *HTSYSMENU*	В системном меню или на кнопке

закрытия дочернего окна

- *HTTOP* — На верхней горизонтальной границе окна
- *HTTOPLEFT* — В левом верхнем углу границы окна
- *HTTOPRIGHT* — В правом верхнем углу границы окна
- *HTTRANSPARENT* — На окне, которое в данный момент перекрыто другим окном
- *HTVSCROLL* — На вертикальной полосе прокрутки
- *HTZOOM* — На значке «Развернуть»

Виртуальный ввод и локальное состояние ввода

При запуске операционная система создает особый поток необработанного ввода и системную очередь аппаратного ввода. Поток ввода бездействует, пока в очереди нет ни одного элемента. Когда пользователь нажимает кнопку мыши или перемещает курсор, либо давит на клавишу, драйвер устройства добавляет аппаратное событие в системную очередь. Поток необработанного ввода активизируется, извлекает из очереди элемент и преобразует его в сообщение, в затем ставит в хвост очереди виртуального ввода конкретного потока. После этого поток ввода ждет появления следующего элемента в системной очереди. Именно системный поток необработанного ввода отвечает за обработку в ОС Windows особых комбинаций клавиш: *Ctrl+Alt+Del*, *Alt+Esc*, *Alt+Tab*. Ни один из других потоков, в том числе пользовательских, не в состоянии перехватить эти комбинации клавиш.

Помимо виртуального ввода потока в ОС Windows поддерживается также концепция локального состояния ввода потока, под которой понимается: для мыши – окно, захватившее мышь, форма курсора мыши, видимость курсора; для клавиатуры – нажатые клавиши, активное окно, окно в фокусе клавиатуры, состояние курсора.

Для того чтобы вывести окно на передний план и подключить его ввод к системному потоку ввода достаточно вызвать функцию

```
#include <windows.h>
// Функция перевода окна на передний план и подключения к системному потоку ввода
//
// FUNCTION: BOOL SetForegroundWindow(HWND)
//
// PARAMETERS:  [in]  hWindow    - хэндл переводимого на передний план окна
//
// RETURN VALUE: ненулевое значение — в случае успеха
//               0                  - в случае неудачи
//
// COMMENTS:  нет
//
BOOL SetForegroundWindow(HWND hWindow);
```

Для получения дескриптора окна, находящегося на переднем плане

используется следующая функция.

```
#include <windows.h>
// Функция получения окна, находящегося на переднем плане
//
// FUNCTION: HWND GetForegroundWindow(VOID)
//
// PARAMETERS:  нет
//
// RETURN VALUE: хэндл окна, находящегося на переднем плане
//
// COMMENTS:  нет
//
HWND GetForegroundWindow(VOID);
```

В некоторых (немногочисленных) приложениях случается, что необходимо, чтобы два потока (или более) совместно использовали одну и ту же виртуальную очередь ввода и локальное состояние ввода. Заставить их делать это можно при помощи функции

```
#include <windows.h>
// Функция подключения к виртуальной очереди ввода другого потока
//
// FUNCTION: BOOL AttachThreadInput(DWORD, DWORD, BOOL)
//
// PARAMETERS:  [in]  dwAttachId   - идентификатор потока, чья очередь не нужна
//              [in]  dwAttachToId - идентификатор потока,
//                                   чья очередь будет использоваться
//              [in]  bAttach      - начать совместное использование или прекратить
//
// RETURN VALUE: ненулевое значение - в случае успеха
//               0                  - в случае неудачи
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//                нужно вызвать GetLastError
//
BOOL AttachThreadInput(DWORD dwAttachId, DWORD dwAttachToId, BOOL bAttach);
```

При этом локальные очереди сообщений остаются индивидуальными. Совместно, повторимся, используется только виртуальная очередь ввода. Это может быть опасно тем, что один из потоков, обрабатывающих нажатие какой-то клавиши, имеет неосторожность зависнуть, тогда другие потоки вообще не получают никакого ввода. Следующий фрагмент кода иллюстрирует использование некоторых из описанных функций.

```
    HWND firstWindow;        // Дескриптор первого окна
    DWORD currentThread;     // Идентификатор текущего потока
    DWORD windowThread;      // Идентификатор потока с окном

    currentThread = GetCurrentThreadId();
    windowThread = GetWindowThreadProcessId(GetForegroundWindow(), NULL);

    AttachThreadInput(currentThread, TRUE);
```

```
    SetForegroundWindow(firstWindow);
```

Управление окнами

Следующая функция позволяет получить экранные координаты указанного окна.

```
#include <windows.h>
// Функция получения экранных координат указанного окна
//
// FUNCTION: BOOL GetWindowRect(HWND, LPRECT)
//
// PARAMETERS:  [in]  hWindow     - идентификатор окна
//              [out] lpRectangle - адрес структуры с координатами окна
//
// RETURN VALUE: ненулевое значение - в случае успеха
//               0                  - в случае неудачи
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//                нужно вызвать GetLastError
//
BOOL GetWindowRect(HWND hWindow, LPRECT lpRectangle);
```

Окно в общем случае состоит из нескольких частей. Имеется возможность получить координаты клиентской части окна, при этом координаты левой верхней точки всегда равны 0.

```
#include <windows.h>
// Функция получения экранных координат клиентской части указанного окна
//
// FUNCTION: BOOL GetClientRect(HWND, LPRECT)
//
// PARAMETERS:  [in]  hWindow     - идентификатор окна
//              [out] lpRectangle - адрес структуры с координатами клиентской части
//
// RETURN VALUE: ненулевое значение - в случае успеха
//               0                  - в случае неудачи
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//                нужно вызвать GetLastError
//
BOOL GetClientRect(HWND hWindow, LPRECT lpRectangle);
```

Структура *RECT* определяет координаты левой верхней и правой нижней точек прямоугольника.

```
#include <windows.h>
typedef struct _RECT
{
    LONG left;
    LONG top;
    LONG right;
    LONG bottom;
} RECT;
```

Следующая функция определяет, принадлежит ли данная точка определенному

прямоугольнику. Точка считается принадлежащей прямоугольнику, если она лежит внутри его четырех сторон либо, если она располагается на левой или верхней сторонах. Если же точка находится на правой или нижней сторонах, то она считается не принадлежащей прямоугольнику.

```
#include <windows.h>
// Функция проверки принадлежности заданной точки к указанной области
//
// FUNCTION: BOOL PtInRect(CONST RECT*, POINT)
//
// PARAMETERS:  [in]  lpRectangle - адрес структуры-прямоугольника
//              [in]  aPoint - структура с точкой
//
// RETURN VALUE: ненулевое значение — точка принадлежит прямоугольнику
//               0                  - точка не принадлежит прямоугольнику
//
// COMMENTS:  Прямоугольник должен содержать нормализованные значения
//
BOOL PtInRect(CONST RECT* lpRectangle, POINT aPoint);
```

См. также http://msdn.microsoft.com/en-us/library/dd162899(VS.85).aspx.

Структура с точкой определена как

```
typedef struct tagPOINT
{
    LONG x;
    LONG y;
} POINT;
```

Экранные координаты преобразуются в клиентские функцией

```
#include <windows.h>
// Функция проверки принадлежности заданной точки к указанной области
//
// FUNCTION: BOOL ScreenToClient(HWND, LPPOINT)
//
// PARAMETERS:  [in]  hWindow - окно с исходными координатами
//              [in]  lpPoint - адрес структуры, содержащей координаты
//
// RETURN VALUE: ненулевое значение — в случае успеха
//               0                  - в случае неудачи
//
// COMMENTS:  нет
//
BOOL ScreenToClient(HWND hWindow, LPPOINT lpPoint);
```

См. также http://msdn.microsoft.com/en-us/library/dd183475(VS.85).aspx.

Для того чтобы получить экранные координаты курсора мыши необходимо вызывать функцию

```
#include <windows.h>
```

```
// Функция получения координат курсора мыши
//
// FUNCTION: BOOL GetCursorPos(HWND, LPPOINT)
//
// PARAMETERS:  [out] lpPoint - адрес структуры с координатами курсора
//
// RETURN VALUE: ненулевое значение — в случае успеха
//               0                  - в случае неудачи
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//                нужно вызвать GetLastError
//
BOOL GetCursorPos(LPPOINT lpPoint);
```

Существует и парная ей функция, которая устанавливает курсор в заданную позицию.

```
#include <windows.h>
// Функция помещения курсора в точку с заданными координатами
//
// FUNCTION: BOOL SetCursorPos(int, int)
//
// PARAMETERS:  [in]  xPosition – горизонтальная позиция
//              [in]  yPosition – вертикальная позиция
//
// RETURN VALUE: ненулевое значение — в случае успеха
//               0                  - в случае неудачи
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//                нужно вызвать GetLastError
//
BOOL SetCursorPos(int xPosition, int yPosition);
```

См. также http://msdn.microsoft.com/en-us/library/ms646970(VS.85).aspx.

Изменение некоторых параметров окна производится функцией.

```
#include <windows.h>
// Функция перемещения окна
//
// FUNCTION: BOOL MoveWindow(HWND, int, int, int, int, BOOL)
//
// PARAMETERS:  [in]  hWindow   – хэндл перемещаемого окна
//              [in]  xPosition – горизонтальная позиция
//              [in]  yPosition – вертикальная позиция
//              [in]  nWidth    – новая ширина окна
//              [in]  nHeight   – новая высота окна
//              [in]  bRepaint  – будет ли окно перерисовываться (TRUE)
//
// RETURN VALUE: ненулевое значение — в случае успеха
//               0                  - в случае неудачи
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//                нужно вызвать GetLastError
//
BOOL MoveWindow(HWND hWindow
```

```
                , int xPosition, int yPosition
                , int nWidth, int nHeight
                , BOOL bRepaint
               );
```

См. также http://msdn.microsoft.com/en-us/library/ff468925(VS.85).aspx.

Для получения и установки параметров полосы прокрутки можно воспользоваться следующей парой функций.

```
#include <windows.h>
// Функция получения параметров полосы прокрутки
//
// FUNCTION: BOOL GetScrollInfo(HWND, int, LPSCROLLINFO)
//
// PARAMETERS:  [in]     hWindow      - описатель окна с полосой прокрутки
//              [in]     barType      - тип полосы прокрутки
//                                      (SB_HORZ -горизонтальная,
//                                      SB_VERT - вертикальная)
//              [in out] lpScrollInfo - указатель на структуру
//                                      с параметрами скроллинга
//
// RETURN VALUE: ненулевое значение — если удалось получить любые параметры
//               0                  - если не удалось получить параметры
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//                нужно вызвать GetLastError
//
BOOL GetScrollInfo(HWND hWindow, int barType, LPSCROLLINFO lpScrollInfo);

// Функция установки параметров полосы прокрутки
//
// FUNCTION: int SetScrollInfo(HWND, int, LPSCROLLINFO, BOOL)
//
// PARAMETERS:  [in]  hWindow      - описатель окна с полосой прокрутки
//              [in]  barType      - тип полосы прокрутки
//                                   (SB_HORZ -горизонтальная,
//                                   SB_VERT - вертикальная)
//              [in]  lpScrollInfo - указатель на структуру
//                                   с параметрами скроллинга
//              [in]  bRedraw      - перерисовывать (TRUE) или нет полосу прокрутки
//
// RETURN VALUE: координаты полосы прокрутки
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//                нужно вызвать GetLastError
//
int SetScrollInfo(HWND hWindow
                , int barType
                , LPSCROLLINFO lpScrollInfo
                , BOOL bRedraw
               );
```

Следующая функция позволяет прокручивать содержимое клиентской области окна, однако единицы прокрутки зависят от устройства.

```
#include <windows.h>
// Функция прокрутки всей или части клиентской области окна
//
// FUNCTION: BOOL ScrollWindowEx(HWND
                                 , int, int, CONST RECT*, CONST RECT*
                                 , HRGN, LPRECT, UINT
                                )
//
// PARAMETERS:  [in]  hWindow      - описатель окна для прокрутки клиентской части
//              [in]  dX           - горизонтальная величина скроллинга.
//                                   Отрицательное значение – «крутить» влево
//              [in]  dY           - вертикальная величина скроллинга.
//                                   Отрицательная – «крутить» вверх
//              [in]  rectScroll   - структура с прямоугольником прокрутки,
//                                   TRUE-прокручивать все
//              [in]  rectClip     - адрес структуры с прямоугольником-клипом
//              [in]  regionUpdate - регион для сохранения непрокрученной части
//              [out] lpUpdate     - обычно NULL
//              [in]  flagScroll   - флаги управления скроллингом (обычно SW_ERASE)
//
// RETURN VALUE: SIMPLEREGION, COMPLEXREGION, NULLREGION — в случае успеха
//               ERROR                                    — в случае ошибки
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//                нужно вызвать GetLastError
//
int ScrollWindowEx(HWND hWindow
                   , int dX, int dY
                   , CONST RECT* rectScroll, CONST RECT* rectClip
                   , HRGN regionUpdate, LPRECT lpUpdate
                   , UINT flagScroll
                  );
```

Еще одна группа функций позволяет направлять и сбрасывать «мышиные» сообщения, связанные с конкретными окнами.

```
#include <windows.h>
// Установка захвата окном сообщений, «приходящих» от мыши
//
// FUNCTION: HWND SetCapture(HWND)
//
// PARAMETERS:  [in]  hWindow      - описатель окна, перехватывающего сообщения
//
// RETURN VALUE: хэндл предыдущего окна, перехватывавшего сообщения или NULL
//
// COMMENTS:  нет
//
HWND SetCapture(HWND hWindow);

// получение идентификатора окна, перехватывающего сообщения
//
// FUNCTION: HWND GetCapture(VOID)
//
// PARAMETERS:  нет
//
// RETURN VALUE: хэндл окна, перехватывавшего сообщения — в случае успеха
```

```
//               NULL — если ни одно из окон не захватывало мышь
//
// COMMENTS:  нет
//
HWND GetCapture(VOID);

// сброс перехвата сообщений
//
// FUNCTION: BOOL ReleaseCapture(VOID)
//
// PARAMETERS:  нет
//
// RETURN VALUE: ненулевое значение — в случае успеха
//               0                  - в случае неудачи
//
// COMMENTS:  Для получения дополнительной информации об ошибке
//                нужно вызвать GetLastError
//
BOOL ReleaseCapture(VOID);
```

Приложение А. Варианты заданий

Вариант 1. Разработать программу, которая демонстрирует эффект «убегания окна от курсора» при его попадании на клиентскую область окна. Завершение «убегания» достигается с помощью двойного щелчка кнопки мыши.

Вариант 2. Разработать программу, которая демонстрирует эффект «прилипания окна к курсору» при его попадании на клиентскую область окна. «Отлипание окна от курсора» производится последовательным перемещением курсора вверх, а потом вниз.

Вариант 3. Разработать программу, которая демонстрирует эффект «убегания окна от курсора» при его попадании на неклиентскую область окна. Завершение «убегания» достигается с помощью двойного щелчка кнопки мыши.

Вариант 4. Разработать программу, которая демонстрирует эффект «прилипания окна к курсору» при его попадании на заголовок окна. Дальнейшее перемещение влево-вправо блокируется, окно может перемещаться только вверх и вниз. «Отлипание окна от курсора» производится двойным щелчком кнопки мыши.

Вариант 5. Разработать программу, которая демонстрирует эффект «прилипания окна к курсору» при его попадании на заголовок окна. Дальнейшее перемещение вверх-вниз блокируется, окно может перемещаться только влево и вправо. «Отлипание окна от курсора» производится двойным щелчком кнопки мыши.

Вариант 6. Разработать программу, которая создает окно с горизонтальной и вертикальной полосами прокрутки. При щелчке кнопки мыши по заголовку окна блокируется вертикальная прокрутка. Блокировка снимается при щелчке кнопки мыши по вертикальной полосе.

Вариант 7. Разработать программу, которая создает окно с горизонтальной и вертикальной полосами прокрутки. При щелчке кнопки мыши по заголовку окна блокируется горизонтальная прокрутка. Блокировка снимается при щелчке кнопки мыши по горизонтальной полосе.

Вариант 8. Разработать программу, которая создает одно окно. Закрытие окна должно выполняться щелчком кнопки по значку сворачивания, при этом должно создаваться новое окно с таким же стилем. Однократный щелчок кнопки по значку закрытия не приводит к выполнению этого действия. Двойной щелчок по этому значку завершает работу программы.

Вариант 9. Разработать программу, которая создает два окна. Действия по сворачиванию, разворачиванию и закрытию одного окна должны выполняться над обоими окнами.

Вариант 10. Разработать программу, которая демонстрирует эффект «прилипания окна к курсору» при его попадании и последующем щелчке кнопки мыши по какой-либо из полос прокрутки. «Отлипание окна от курсора» производится трехкратным щелчком кнопки мыши.

Вариант 11. Разработать программу, которая демонстрирует эффект «прилипания окна к курсору» при его попадании и последующем щелчке кнопки мыши по заголовку окна. «Отлипание окна от курсора» производится трехкратным щелчком кнопки мыши.

Вариант 12. Разработать программу, которая создает окно с горизонтальной и вертикальной полосами прокрутки. При щелчке кнопки мыши по вертикальной полосе прокрутки окна блокируется горизонтальная прокрутка. Блокировка снимается при повторном щелчке кнопки мыши по вертикальной полосе.

Вариант 13. Разработать программу, которая создает окно с горизонтальной и вертикальной полосами прокрутки. При щелчке кнопки мыши по горизонтальной полосе прокрутки окна блокируется вертикальная прокрутка. Блокировка снимается при щелчке кнопки мыши по горизонтальной полосе.

Вариант 14. Разработать программу, которая создает одно окно. Закрытие окна должно выполняться щелчком кнопки по значку разворачивания окна, при этом должно создаваться новое окно с таким же стилем. Однократный щелчок кнопки по значку закрытия не приводит к выполнению этого действия. Двойной щелчок по этому значку завершает работу программы.

Вариант 15. Разработать программу, которая создает три окна. Действия по сворачиванию и разворачиванию любого из них должны выполняться над всеми окнами.

Вариант 16. Разработать программу, которая демонстрирует эффект «прилипания окна к курсору» при его попадании и последующем двойном щелчке кнопки мыши по какой-либо из полос прокрутки. «Отлипание окна от курсора» производится повторным двойным щелчком кнопки мыши по полосе прокрутки.

Вариант 17. Разработать программу, которая демонстрирует эффект «прилипания окна к курсору» при его попадании и последующем двойном щелчке кнопки мыши по заголовку окна. «Отлипание окна от курсора» производится повторным двойным щелчком кнопки мыши по заголовку окна.

Вариант 18. Разработать программу, которая создает одно окно. Закрытие окна должно выполняться щелчком кнопки по значку закрытия окна, при этом должно создаваться новое окно с таким же стилем. Двойной щелчок по этому значку завершает работу программы.

Вариант 19. Разработать программу, которая создает три окна. Действия по сворачиванию и закрытию любого из них должны выполняться над всеми окнами.

Вариант 20. Разработать программу, которая демонстрирует эффект «прилипания окна к курсору» при его попадании и последующем щелчке кнопки мыши по вертикальной полосе прокрутки. «Отлипание окна от курсора» производится повторным щелчком кнопки мыши по этой же полосе прокрутки.

Вариант 21. Разработать программу, которая демонстрирует эффект «прилипания окна к курсору» при его попадании и последующем щелчке кнопки мыши по горизонтальной полосе прокрутки. «Отлипание окна от курсора» производится последовательным перемещением курсора влево-вправо-вверх.

Вариант 22. Разработать программу, которая создает три окна с заголовками «Window_1», «Window_2» и «Window_3». Щелчок кнопки мыши по значку закрытия

«Window_1» закрывает только «Window_2» и «Window_3», а щелчок по такому же значку на «Window_2» и «Window_3» приводит к завершению работы программы.

Вариант 23. Разработать программу, которая создает три окна. Действия по разворачиванию и закрытию любого из них должны выполняться над всеми окнами.

Вариант 24. Разработать программу, которая демонстрирует эффект «прилипания окна к курсору» при его попадании и последующем щелчке управляющей кнопки мыши по системному меню. «Отлипание окна от курсора» производится повторным щелчком управляющей кнопки мыши.

Вариант 25. Разработать программу, которая демонстрирует эффект «прилипания окна к курсору» при его попадании и последующем щелчке вспомогательной кнопки мыши по системному меню. «Отлипание окна от курсора» производится последовательным перемещением курсора вправо-влево-вниз.

Вариант 26. Разработать программу, в которой при попадании курсора мыши на клиентскую область окна и последующей попытке его вывода за нее, окно движется в том же направлении (эффект прилипания окна к курсору, но только если курсор пытаются вывести за клиентскую область окна). «Отлипание» происходит при нажатии, кнопки созданной в клиентской области окна.

Вариант 27. Разработать программу, которая демонстрирует эффект «прозрачной клиентской области», т. е. сквозь окно должен быть виден рабочий стол (видимость других окон обеспечивать необязательно). Совет: можно воспользоваться «многослойными окнами». См. «Layered Windows» — http://msdn.microsoft.com/en-us/library/ms997507.aspx, а также «window with transparent client area» — http://stackoverflow.com/questions/14043921/window-with-transparent-client-area.

Вариант 28. Разработать программу, которая создаёт окно. При каждом щелчке на клиентской области окна должна создаваться новая кнопка. Нажатие любой из созданных кнопок должно завершать работу программы.

Вариант 29. Разработать программу «рисования» дочерних окон на родительском окне. Рисования производится следующим образом:

1. Активация режима рисования по нажатию левой кнопки мыши (сохранение первой пары координат окна).
2. Растягивание элемента управления в соответствии с перемещением указателя мыши (формирование второй пары координат окна).

3. Деактивация режима рисования по нажатию правой кнопки мыши (сохранение второй пары координат окна).

Класс окна для рисования изменяется по нажатию на клавиатуре клавиши, соответствующей первой букве в названии класса. Рисуются окна следующих классов: BUTTON, COMBOBOX, EDIT, LISTBOX, SCROLLBAR.

Совет: см. «Keyboard Input» — http://msdn.microsoft.com/en-us/library/windows/desktop/ms645530(v=vs.85).aspx, а также «Работа с клавиатурой» - http://www.sources.ru/msdn/library/using_keyboard_input.shtml.

Вариант 30. Разработать программу, которая создает три окна, клиентские области которых, соответственно, имеют цвета: зеленый, синий, красный. При щелчке на каком-либо из этих окон должна происходить перестановка цветов в случайном порядке, при двойном щелчке все три окна перекрашиваются в цвет того окна, на котором был произведен двойной клик. Совет: можно воспользоваться BeginPaint (http://msdn.microsoft.com/en-us/library/windows/desktop/dd183362(v=vs.85).aspx) и EndPaint (http://msdn.microsoft.com/en-us/library/windows/desktop/dd162598(v=vs.85).aspx)

Вариант 31. Разработать программу, позволяющую осуществлять горизонтальный скроллинг содержимого окна с помощью вертикальной полосы прокрутки.

Вариант 32. Разработать программу, позволяющую осуществлять вертикальный скроллинг содержимого окна с помощью горизонтальной полосы прокрутки.