Урок 2. Стили классов окон, оконные стили, управление окнами

На базе одного класса окна приложение может создать несколько окон. Все эти окна могут быть сделаны в одном или нескольких стилях.

- Стиль окна определяет внешний вид окна и его поведение.
- Для класса окна также определяется понятие стиля - стиль класса определяет внешний вид и поведение **всех** окон, созданных на базе данного класса.

Стиль класса окна

Стиль класса окна определяется при регистрации класса окна. Стиль класса окна задается в виде отдельных битов (разрядов), для которых определены символические константы с префиксом CS_. Определенный в классе окна стиль класса окна используется при создании всех окон на базе этого класса.

Чаще всего используется стиль CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS. Как уже говорилось, если для класса заданы стили CS_HREDRAW и CS_VREDRAW, то при изменении размеров окна функция окна может получить сообщение WM_PAINT. В этом случае функция окна должна перерисовать часть окна или все окно.

Стиль CS_DBLCLKS используется при необходимости отслеживать двойные щелчки мышью. При этом в функцию окна посылаются сообщения WM_LBUTTONDBLCLK и WM_RBUTTONDBLCLK.

Если этот стиль не будет задан, функция окна получит только идущие парами сообщения об одиночном нажатии клавиш мыши WM_LBUTTONDOWN и WM_RBUTTONDOWN.

Стили окон, окна основных стилей

Определенный в классе окна стиль класса окна используется при создании всех окон на базе этого класса. Для дальнейшего уточнения внешнего вида и поведения окна используется другая характеристика - **стиль окна**. Стиль окна указывается при создании окна функцией CreateWindow (CreateWindowEx). Для определения стиля окна используются символические константы с префиксом WS_.

Рассмотрим чаще всего используемые основные стили: **перекрывающиеся** окна (overlapped window), **всплывающие** (или временные, или выпадающие) окна (pop-up window), **дочерние** окна (child window).

Перекрывающиеся окна

Перекрывающиеся окна имеют заголовок (title bar), рамку и внутреннюю часть окна (client region). Дополнительно перекрывающиеся окна могут иметь ***системное меню***, ***кнопки*** для максимального увеличения размера окна и для свертки окна в пиктограмму, вертикальную и горизонтальную ***полосу просмотра и меню***.

Перекрывающиеся окна обычно используются в качестве ***главного*** окна приложения. Для определения стиля перекрывающегося окна существует символическая константа WS_OVERLAPPEDWINDOW, определенная как поразрядное ИЛИ нескольких констант:

```
#define WS_OVERLAPPEDWINDOW (WS_OVERLAPPED| \
```

```
WS_CAPTION | WS_SYSMENU | WS_THICKFRAME| \
WS_MINIMIZEBOX | WS_MAXIMIZEBOX)
```

Приложение Windows может создавать несколько перекрывающихся окон. При создании окон при помощи функции CreateWindow в качестве восьмого параметра функции можно указать дескриптор окна-владельца (переменная типа HWND). Окно-владелец уже должно существовать на момент создания окна, имеющего владельца.

Перечислим особенности перекрывающихся окон.

- Если окно-хозяин сворачивается в пиктограмму, все окна, которыми оно владеет, становятся невидимыми.
- Если сначала свернули в пиктограмму окно, которым владеет другое окно, а затем и окно-хозяина, то пиктограмма подчиненного окна исчезнет.
- При уничтожении окна-владельца автоматически уничтожаются и все принадлежащие ему окна.
- Обычное перекрывающееся окно, не имеющее окна-владельца, может располагаться в любом месте экрана и принимать любые размеры. Подчиненные окна располагаются всегда над поверхностью окна-владельца, перекрывая его изображение.
- Координаты создаваемых функцией CreateWindow перекрывающихся окон указываются по отношению ко всему экрану, т.е. при создании окна с координатами (0,0), оно будет расположено в ***верхнем левом углу*** экрана.
- При изменении размеров перекрывающегося окна функция окна получает сообщение WM_SIZE, в параметрах которого указаны новые размеры окна.

Временные окна

Другим базовым стилем является стиль ***временных*** окон, которые обычно используются для вывода информационных сообщений и остаются на экране непродолжительное время. Временные окна, в отличие от перекрывающихся, могут не иметь заголовка. Если для временного меню заголовок определен, тогда оно может иметь и системное меню.

Часто для создания выпадающих окон, имеющих рамку, используется стиль WS_POPUPWINDOW, определенный как поразрядное ИЛИ нескольких констант (для того чтобы к временному окну добавить системное меню и заголовок, следует использовать комбинацию WS_POPUPWINDOW | WS_CAPTION):

```
#define WS_POPUPWINDOW (WS_POPUP | WS_BORDER | WS_SYSMENU)
```

Особенности вр*е*менных окон.

- Вр*е*менные окна могут иметь окно-владельца и могут сами владеть другими окнами. Замечания относительно владения перекрывающимися окнами, справедливы и для временных окон.
- Начало системы координат, используемой при создании временных окон, находится в левом верхнем углу экрана. Поэтому при создании вр*е*менных окон применяются экранные координаты, как и при создании перекрывающихся окон.
- При изменении размеров вр*е*менного окна функция окна (WndProc) получает сообщение WM_SIZE, в параметрах которого указаны ***новые размеры*** окна.

Дочерние окна

Дочерние окна чаще всего используются приложениями Windows. Эти окна нужны для создания органов управления, таких, как кнопки (BUTTON) или списки (например, LISTBOX). Все органы управления - дочерние окна.

Стиль дочернего окна определяется константой WS_CHILDWINDOW. В отличие от перекрывающихся и временных окон дочерние окна, как правило, не имеют рамки, заголовка, кнопок минимизации и максимального увеличения размеров окна, а также полос просмотра. Дочерние окна ***сами рисуют*** все, что в них должно быть изображено.

Особенности дочерних окон.

- Дочерние окна должны иметь окно-родителя. Только дочерние окна могут иметь родителей, перекрывающиеся и временные окна могут иметь окно-хозяина, но не родителя. У дочерних окон могут быть “братья” (или “сестры”).
- Дочерние окна всегда располагаются ***на поверхности*** окна-родителя.
- При создании дочернего окна начало системы координат расположено в левом верхнем углу внутренней поверхности окна-родителя (но не в верхнем углу экрана, как для перекрывающихся и временных окон).
- Так как дочернее окно как бы “прилипает” к поверхности окна-родителя, то при щелчке мышью над поверхностью дочернего окна, сообщение от мыши попадет в функцию дочернего, а не родительского окна.
- При создании дочернего окна в качестве девятого параметра (вместо идентификатора меню, которого не может быть у дочернего окна) функции CreateWindow необходимо указать идентификатор дочернего окна (переменная типа int, любое целое число).
- Если для приложения создается несколько дочерних окон, для каждого окна необходимо указать свой идентификатор (ID_), эти идентификаторы будут использоваться дочерними окнами при отправлении сообщений (WM_) родительскому окну.
- Дочернее окно “прилипает” к поверхности родительского окна и перемещается вместе с ним. Оно никогда не может выйти за пределы родительского окна.
- ***Все*** дочерние окна ***скрываются при сворачивании окна-родителя*** в пиктограмму и появляются вновь при восстановлении родительского окна.
- При изменении размеров родительского окна дочерние окна получают сообщение WM_PAINT, но не получают сообщения WM_SIZE, это сообщение попадает только в родительское окно.

Пример создания окон различных стилей

Фрагмент функции WinMain, который создает окна разных стилей на основе двух классов окон:

```
// регистрация классов окон с именами "MainWindow”, "Window1", "Window2"
. . .
// создание главного перекрывающегося окна
HWND hWndMain = CreateWindow("MainWindow", "OVERLAPPEDWINDOW",
        WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, hInstance, NULL);
if(hWndMain == 0) return FALSE;
ShowWindow(hWndMain, nCmdShow);
UpdateWindow(hWndMain);
```

```
// создание временного окна
HWND hWndPopup = CreateWindow("Window1","POPUPWINDOW",
WS_POPUPWINDOW | WS_CAPTION | WS_MINIMIZEBOX| WS_MAXIMIZEBOX|WS_VISIBLE,
100, 100, 300, 300, hWndMain, NULL, hInstance, NULL);
if(hWndPopup == 0)
{ DestroyWindow(hWndMain);
return FALSE; }
ShowWindow(hWndPopup, nCmdShow);
UpdateWindow( hWndPopup );

// создание дочернего окна
HWND hWndChild = CreateWindow("Window2", "CHILDWINDOW",
WS_CHILDWINDOW | WS_CAPTION | WS_MINIMIZEBOX | WS_MAXIMIZEBOX,
150, 150, 250, 250, hWndMain, NULL, hInstance, NULL);
if(hWndChild == 0)
{ DestroyWindow( hWndMain );
DestroyWindow( hWndPopup );
return FALSE; }
ShowWindow(hWndChild, nCmdShow);
UpdateWindow(hWndChild);
// цикл обработки очереди сообщений
. . .
```

Окна с полосами прокрутки

Для того чтобы добавить в окно приложения вертикальную или горизонтальную полосу прокрутки необходимо включить стиль WS_VSCROLL и WS_HSCROLL или оба сразу в описание стиля создаваемого функцией CreateWindow окна.

Особенности:

- Рабочая область окна не включает в себя пространство, занятое полосами прокрутки. Ширина вертикальной полосы и высота горизонтальной полос прокрутки постоянны для конкретного дисплейного драйвера (эти значения можно получить с помощью функции GetSystemMetrics).
- ОС Windows обеспечивает всю логику работы мыши с полосами прокрутки. Однако у полос прокрутки нет интерфейса клавиатуры. Если приложение желает дублировать клавишами управления курсором некоторые функции полос прокрутки, оно должно самостоятельно реализовать эту логику при обработке клавиатурных сообщений.

Диапазон и положение полос прокрутки

Каждая полоса прокрутки имеет соответствующий диапазон (range) – два целых числа, отражающих минимальное и максимальное значение, и положение (position) – местоположение бегунка внутри диапазона. По умолчанию, устанавливается следующий диапазон прокрутки – минимум 0 и максимум 100, но диапазон легко изменить на любое другое значение с помощью функции SetScrollRange (для полос прокрутки окна hWnd):

```
SetScrollRange(hWnd, iBar, iNewMin, iNewMax, bRedraw);
```

Параметр iBar равен либо SB_VERT, либо SB_HORZ, iNewMin и iNewMax являются минимальной и максимальной границами диапазона, а bRedraw устанавливается в

TRUE, если необходимо, чтобы Windows перерисовала полосы в соответствии с новыми значениями.
Положение бегунка всегда дискретно. Например, полоса прокрутки с диапазоном от 0 до 4 имеет пять положений бегунка. Для установки нового положения бегунка можно использовать функцию SetScrollPos:

```
SetScrollPos(hWnd, iBar, iNewPos, bRedraw);
```

Для определения текущего диапазона полосы прокрутки и текущего положения бегунка используются функции GetScrollRange и GetScrollPos.

Если приложение использует полосы прокрутки, то оно совместно с Windows берет на себя ответственность за поддержку полос прокрутки и обновлению положения бегунка.
Сферы ответственности Windows:

- Управление логикой работы мыши с полосой прокрутки.
- Обеспечение временной “инверсии цвета” при нажатии на кнопку мыши в полосе прокрутки.
- Перемещает бегунок в соответствие с тем, как внутри полосы прокрутки его перемещает пользователь.
- Отправляет сообщения полосы прокрутки в оконную процедуру для окна, содержащего полосу прокрутки.

Сферы ответственности приложения по поддержке полос прокрутки:

- Инициализация диапазона полосы прокрутки.
- Обработка сообщений полосы прокрутки.
- Обновление положения бегунка полосы прокрутки.

Сообщения полос прокрутки

Windows посылает оконной процедуре сообщения WM_VSCROLL и WM_HSCROLL, когда на полосе прокрутки щелкают мышью или перетаскивают бегунок.

Замечание. При работе с оконными полосами прокрутки следует игнорировать параметр lParam, он используется только для полос прокрутки – элементов управления.

Младшее слово параметра wParam этих сообщений – это число, показывающее, что мышь осуществляет какие-то действия на полосе прокрутки. Его значение соответствует определенным идентификаторам, которые начинаются на SB_:

```
int nScrollCode = (int) LOWORD(wParam);      // произведенное действие
short int nPos = (short int) HIWORD(wParam); // текущая позиция
```

- Оконная процедура может получить сообщения с кодами типа SB_LINEUP, SB_PAGEUP, SB_LINEDOWN и SB_PAGEDOWN, если пользователь изменяет текущее положение на 1 положение или на 1 “страницу” (эти коды приходят с сообщениями, как от вертикальной, так и от горизонтальной полос прокрутки).
- Сообщения с кодом SB_ENDSCROLL приходят, когда кнопка мыши отпущена. Как правило, приложения ***игнорируют*** такие сообщения.
- Если пользователь перемещает бегунок при помощи мыши, то сообщение от полосы прокрутки несет с собой в младшем слове параметра wParam код SB_THUMBTRACK (таких сообщений может быть слишком много!). Если затем пользователь отпускает кла-

вишу мыши, то в оконную процедуру поступает сообщение с кодом SB_THUMBPOSITION.

- Если младшее слово параметра wParam равно SB_THUMBTRACK или SB_THUMBPOSITION, то в этих случаях старшее слово wParam определяет текущее положение полосы прокрутки. Во всех остальных случаях работы с полосой прокрутки старшее слово wParam можно игнорировать.

Замечание. Согласно документации, сообщения с кодами SB_TOP и SB_BOTTOM показывают, что полоса прокрутки переведена в свое максимальное или минимальное положение. Однако, если приложение самостоятельно не реализовывает интерфейс для работы с полосой прокрутки при помощи клавиатуры и не посылает само себе такие сообщения по мере необходимости, то тогда оно никогда его не получит от Windows.

Пример обработки сообщений от полосы прокрутки окна

Пусть создано окно hWnd с вертикальной линейкой прокрутки и затем для нее установлены диапазон и текущее положение бегунка:

```
static int min_sb = 1, max_sb = 100, pos_sb = 20;
SetScrollRange(hWnd, SB_VERT, min_sb, max_sb, TRUE);
SetScrollPos(hWnd, SB_VERT, pos_sb, TRUE);
```

Пример оконной функции созданного окна, демонстрирующий возможную обработку действий пользователя с вертикальной линейкой прокрутки

```
. . .
case WM_VSCROLL:
{
        // запоминаем предыдущую позицию бегунка
        int  old_pos_sb = pos_sb;

        // lParam для оконных полос просмотра всегда равен NULL
int  nScrollCode = (int)LOWORD(wParam);       // произведенное действие
short int nPos = (short int)HIWORD(wParam);   // текущая позиция

        // изменеяем текущую позицию бегунка
        switch(nScrollCode)
        {
                case SB_PAGEDOWN:      pos_sb += 10; break;
                case SB_PAGEUP:        pos_sb -= 10; break;
                case SB_LINEDOWN:      pos_sb += 1; break;
                case SB_LINEUP:        pos_sb -= 1; break;
                case SB_THUMBPOSITION: pos_sb = nPos; break;
                case SB_THUMBTRACK:    pos_sb = nPos; break;
                default: return 0;
        }
        if(pos_sb < min_sb)            pos_sb = min_sb;
        else if(pos_sb > max_sb)       pos_sb = max_sb;

        if(old_pos_sb != pos_sb)
        SetScrollPos(hWnd, SB_VERT, pos_sb, TRUE);
}; return 0;
. . .
```

Различные метрики Windows

В программном интерфейсе Windows имеются функции, позволяющие получить информацию о размерах отдельных компонент Windows, таких, как ширина рамки окна, ширина и высота экрана и т.п. Эта информация нужна для определения габаритов и расположения создаваемых окон и других изображений.

Системные метрики

Метрики системных компонент Windows можно определить при помощи функции GetSystemMetrics. Единственный аргумент этой функции задает параметр, значение которого необходимо определить:

```
int GetSystemMetrics(int nIndex);
```

Для определения того или иного компонента в заголовочных файлах Windows имеются константы с префиксом SM_. Рассмотрим некоторые из них:

- SM_CXCURSOR - ширина курсора.
- SM_CXICON - ширина пиктограммы.
- SM_CXSCREEN - ширина экрана.
- SM_CYCAPTION - высота заголовка окна.
- SM_CYCURSOR - высота курсора.
- SM_CYICON - высота пиктограммы.
- SM_CYMENU - высота одной строки в полосе меню.
- SM_CYSCREEN - высота экрана.
- SM_CYHSCROLL – высота горизонтальной полосы прокрутки.
- SM_CXVSCROLL – ширина вертикальной полосы прокрутки.

Пример определения высоты графического экрана:

```
int h = GetSystemMetrics(SM_CYSCREEN);
```

Определение возможностей устройств ввода/вывода

В программном интерфейсе Windows имеется функция, позволяющая по контексту устройства определить возможности и параметры драйверов, обслуживающих устройства ввода/вывода:

```
int GetDeviceCaps(HDC hdc, int iCapability);
```

Первый параметр функции (hdc) задает ***контекст устройства***, для которого необходимо получить информацию о его возможностях (получить контекст устройства можно от функций BeginPaint при обработке сообщения WM_PAINT или GetDC при обработке сообщения, отличного от WM_PAINT).

Второй параметр (`iCapability`) функции GetDeviceCaps определяет параметр устройства, значение которого необходимо получить.

Замечание. Остановимся на некоторых из возможных значений параметров устройства. Значения ASPECTX, ASPECTY, ASPECTXY определяют размеры пиксела. Но, пикселы не всегда квадратные. Поэтому масштаб изображения по оси x может отличаться от масштаба по оси y. Размеры пиксела позволяют вычислить отно-

шение сторон пиксела и выполнить правильное масштабирование. В этом случае отображаемые окружности будут круглыми, а квадраты - квадратными.

Пример получения информации о соотношение размеров пиксела по осям X и Y при обработке сообщения WM_PAINT в некоторой оконной функции:

```
. . .
case  WM_PAINT:
{PAINTSTRUCT ps;
 HDC hDC = BeginPaint(hWnd, &ps); // получение контекста отображения окна hWnd
        int asp_xy = GetDeviceCaps(hDC, ASPECTXY);
        . . .
        // использование asp_xy
        EndPaint(hWnd,&ps);
        };
        return 0;
```

Определение размеров окна

Приложение Windows, как правило, «не знает» заранее размеров отведенных ему окон. Поэтому приложение Windows должно уметь определять размеры своих окон.

Существует **<u>два</u>** разных метода определения размера окна.

Первый метод основан на том факте, что при изменении размера окна функция окна (WndProc) получает сообщение WM_SIZE, параметры которого несут с собой информацию о новых размерах. Параметры сообщения WM_SIZE описывают новый размер окна и способ, которым оно изменило размер:

- wParam - способ, при помощи которого окно изменило свой размер;
- LOWORD(lParam) - новая ширина клиентской части окна;
- HIWORD(lParam) - новая высота клиентской части окна.

Параметр wParam может принимать одно из нескольких значений, символические имена которых начинаются с префикса SIZE_ (например, SIZE_MAXIMIZED – окно было максимизировано).

Обрабатывая сообщение WM_SIZE, функция окна должна сохранить параметры сообщения и послать сама себе сообщение WM_PAINT. Для этого при помощи функции InvalidateRect нужно объявить все окно как недействительное и затем вызвать функцию UpdateWindow. При обработке сообщения WM_PAINT необходимо пользоваться сохраненными значениями новых размеров клиентской части окна.

Второй метод определения размеров окна заключается в вызове функций, возвращающих размеры окна ***по*** его ***идентификатору (ID)***. Например, функция GetClientRect предназначена для определения координат внутренней области окна (эти координаты вычисляются относительно левого верхнего угла внутренней области окна):

```
        void  GetClientRect(HWND hwnd, RECT FAR* lprc);
```

Фрагменты оконной функции, определяющей размеры клиентской части окна двумя способами и осуществляющей вывод в зависимости от этих размеров:

```
        static h, w;
        . . .
                case WM_SIZE:
                {       w = LOWORD( lParam );
```

```
                h = HIWORD( lParam );
                InvalidateRect(hWnd, NULL);
                UpdateWindow( hWnd );
        };
        return 0;
        . . .
        case WM_PAINT:
        {       PAINTSTRUCT ps;
                HDC hDC = BeginPaint(hWnd, &ps);

        // 1-ый способ вывода в соответствии с текущими размерами окна
                . . . // вывод с использованием значений w и h

                EndPaint(hWnd, &ps);
        };
        return 0;
```

или

```
        . . .
        case WM_PAINT:
        {       PAINTSTRUCT ps;
                HDC hDC = BeginPaint(hWnd, &ps);

        // 2-ой способ вывода в соответствии с текущими размерами окна
                Rect r;
                GetClientRect(hWnd, &r);

        . . . // вывод с использованием значений полей структуры r

                EndPaint(hWnd, &ps);
        };
        return 0;
        . . .
```

Определение расположения окна

При перемещении окна функция окна получает сообщение WM_MOVE, вместе с ним она получает новые координаты внутренней области окна:

- wParam - не используется;
- LOWORD(lParam) - X-координата верхнего левого угла клиентской части окна;
- HIWORD(lParam) - Y-координата верхнего левого угла клиентской части окна.

Замечание. Для окон, имеющих стили WS_OVERLAPPED и WS_POPUP, координаты отсчитываются от верхнего левого угла экрана. Для окон стиля WS_CHILD эти координаты отсчитываются от верхнего левого угла внутренней области родительского окна.

В любой момент времени приложение может также определить расположение и размер окна, вызвав функцию:

```
void  GetWindowRect(HWND hwnd, RECT FAR* lprc);
```

Эта функция выдает информацию о расположении и размере прямоугольной области, ограничивающей окно, с учетом заголовка, рамки и полос просмотра. Все координаты отсчитываются от верхнего левого угла экрана.

Метрики текста

Приложения Windows могут выводить текст с использованием различных шрифтов. Все шрифты можно разделить на две группы относительно ширины букв:

- К первой относятся шрифты с ***фиксированной шириной букв (fixed-pitch font)***. Все буквы такого шрифта имеют одинаковую ширину.
- Вторая группа шрифтов - шрифты с ***переменной шириной букв (variable-pitch font)***. Для таких шрифтов каждая буква имеет свою ширину.

Кроме того, шрифты также можно разделить на следующие группы относительно начертания:

- ***Растровые шрифты (raster font)*** состоят из отдельных пикселов. Если выполнить масштабирование растрового шрифта в сторону увеличения размера букв, наклонные линии и закругления будут изображаться в виде “лестницы”.
- ***Контурные шрифты (stroke font)*** больше подходят для плоттеров. При масштабировании таких шрифтов можно достигнуть лучших результатов, чем при масштабировании растровых. Однако при большом размере букв результат все равно получается неудовлетворительный.
- ***Масштабируемые шрифты*** **TrueType** сохраняют начертание символов при любом изменении размеров, поэтому они используются чаще всего.

В контекст отображения по умолчанию выбран так называемый системный шрифт SYSTEM_FONT. Системный шрифт используется в Windows, например, для текста в заголовках окон, меню и диалоговых панелях. Системный шрифт относится к растровым шрифтам с переменной шириной букв.

Переменная ширина букв усложняет задачу вывода текста, но в составе программного интерфейса Windows имеется функция, предназначенная для подсчета длины текстовой строки lpszString длиной cbString

```
DWORD GetTextExtent(HDC hdc, LPSTR lpszString, int cbString);
```

Для получения информации о шрифте, выбранном в контекст устройства, предназначена функция:

```
BOOL GetTextMetrics(HDC hdc, TEXTMETRIC FAR* lptm);
```

Хорошо спроектированное приложение позволяет пользователю выбирать для отображения текста произвольные шрифты. Поэтому приложение никогда не должно ориентироваться на конкретные размеры шрифта. Вместо этого следует определять эти размеры динамически с помощью специально предназначенных для этого средств.

Управление окнами Windows

Окно в Windows можно определить как прямоугольную область на экране. Однако это определение в своей простоте скрывает множество функциональных возможностей под абстрактной идеей окна как основной единицы взаимодействия пользователя и Windows-приложения.

Окна Windows

Окно – это не только область на экране, но еще и адресат событий и сообщений в среде Windows.

- Окно идентифицируется по дескриптору окна. Этот дескриптор (переменная типа HWND) однозначно определяет каждое окно в системе. Список окон содержит очевидные окна приложений и диалоговых панелей, а также менее очевидные, такие, как рабочий стол, пиктограммы или кнопки.
- События пользовательского интерфейса объединяются с дескриптором соответствующего окна, образуя сообщение Windows, и посылаются или помещаются в очередь приложения (точнее, в очередь потока), который владеет этим окном.

Естественно, Win32 API предоставляет множество функций по созданию и управлению окнами.

Иерархия окон

Windows организует свои окна в иерархическую структуру:

- Каждое окно имеет родителя, корнем дерева всех окон является окно рабочего стола, создаваемого Windows при загрузке.
- Для всех окон верхнего уровня (для главных окон приложений и других перекрывающихся и всплывающих окон приложений) родительским окном является рабочий стол.
- Родитель дочернего окна – окно верхнего уровня или другое дочернее окно, более высокого уровня по иерархии.

Между окнами верхнего уровня (перекрывающиеся и всплывающие окна) существует иерархическая связь. Владельцем окна верхнего уровня может быть другое окно того же уровня. Окно, имеющее владельца, всегда отображается поверх своего владельца и исчезает при минимизации окна-владельца.

Типичным случаем владения одного окна верхнего уровня другим является приложение, отображающее диалоговое окно. Диалоговое окно не является дочерним окном (оно является всплывающим окном), но его владельцем остается окно приложения.

Наиболее часто употребляемые функции, позволяющие приложению исследовать иерархию окон и находить определенные окна:

- GetDesktopWindow – позволяет приложению получить дескриптор окна рабочего стола.
- FindWindow – используется для поиска окна верхнего уровня по имени его класса окна или по заголовку окна.
- GetParent – идентифицирует родительское окно указанного окна.
- GetWindow – предоставляет наиболее гибкий способ с иерархией окон. В зависимости от значения второго параметра эту функцию можно использовать для получения идентификатора родительского окна, владельца, окон одного уровня или дочерних окон.

Сообщения управления окнами

Наиболее часто обрабатываемые сообщения:

- WM_CREATE - это первое сообщение, которое получает оконная процедура созданного класса. Оно посылается перед тем, как окно станет видимым, и перед завершением функций CreateWindow или CreatWindowEx. В ответ на это сообщение приложение может выполнить необходимые функции инициализации перед тем, как окно станет видимым.
- WM_DESTROY - посылается оконной процедуре окна, которое уже удалено с экрана и должно быть разрушено.
- WM_CLOSE - указывает, что окно должно быть закрыто. Обработчик по умолчанию в функции DefWindowProc при получении этого сообщения вызывает DestroyWindow. Приложение может, например, вывести диалоговое окно подтверждения и вызвать DestroyWindow только в случае подтверждения пользователем закрытия окна.
- WM_QUIT - это обычно последнее сообщение, которое получает основное окно приложения. Получение этого сообщения приводит к возврату нулевого значения функцией GetMessage, что в свою очередь приводит к завершению цикла сообщений большинства приложений. Это сообщение требует завершить приложение. Оно генерируется в ответ на вызов функции PostQuitMessage.
- WM_QUERYENDSESSION - уведомляет приложение о намерении Windows закончить сеанс. Приложение может возвратить значение FALSE в ответе на это сообщение, предотвратив этим выключение Windows. После обработки сообщения WM_QUERYENDSESSION Windows посылает всем приложениям сообщение WM_ENDSESSION с результатами этой обработки.
- WM_ENDSESSION - посылается приложениям после обработки сообщения WM_QUERYENDSESSION. Оно указывает, должна ли Windows выключиться, или выключение отложено. При указании выключения сеанс Windows может закончиться в любое время после обработки сообщения WM_ENDSESSION всеми приложениями. Поэтому важно, чтобы приложения выполнили все задачи по обеспечению безопасного завершения работы.
- WM_ACTIVATE - указывает, что окно верхнего уровня будет активизировано или деактивизировано. Сообщение сначала посылается окну, которое должно быть деактивизировано, а потом окну, которое должно быть активизировано.
- WM_SHOWWINDOW - указывает, что окно должно быть скрыто или отображено. Окно может быть скрыто в результате вызова функции ShowWindow или в результате максимизации другого окна.
- WM_ENABLE – посылается окну, когда оно становится доступным или недоступным. Это может произойти после вызова функции EnableWindow. Недоступное окно не может принимать вводимые данные от мыши или клавиатуры.
- WM_MOVE – указывает, что расположение окна изменилось.
- WM_SIZE – указывает,что размер окна был изменен.
- WM_SETFOCUS – указывает получение окном фокуса клавиатуры.
- WM_KILLFOCUS – указывает, что окно должно потерять фокус клавиатуры.
- WM_GETTEXT – посылается окну, запрашивая копирование текста окна в буфер. Для большинства окон текстом является их заголовок. Для элементов управления типа

кнопок и т.д. текстом окна является текст, отображаемый в этих элементах. Обычно это сообщение обрабатывается функцией DefWindowProc.

- WM_SETTEXT – запрашивает установку текста окна из содержимого буфера. Функция DefWindowProc в ответ на это сообщение устанавливает текст окна и соответственно его отображает.

Оконная процедура, стандартные оконные процедуры

Каждое окно связано с классом окна. Класс окна – это класс, или предоставляемый Windows (заранее определенный класс), или определенный пользователем и зарегистрированный с помощью функции RegisterClass.

Задача класса окна – определение характеристик и функциональных возможностей множества связанных с ним окон.

Наиболее примечательным, но, разумеется, не единственным свойством класса окна является оконная процедура. Именно через оконную процедуру реализуется поведение окна. Отвечая на различные сообщения, оконная процедура определяет реакцию окна на события мыши и клавиатуры и изменение его внешнего вида в соответствии с этими событиями.

Оконная процедура вызывается каждый раз, когда сообщение посылается окну с помощью функции SendMessage, и каждый раз, когда сообщение отправляется окну посредством функции DispatchMessage. Роль оконной процедуры заключается в обработке посланных и отправленных этому окну сообщений, при этом она может пользоваться оконной процедурой по умолчанию для обработки не интересующих ее сообщений.
Windows поддерживает две оконные процедуры по умолчанию: DefWindowProc и DefDlgProc.

- Функция DefWindowProc реализует поведение по умолчанию типичного окна верхнего уровня.
- Функция DefDlgProc используется диалоговыми окнами, и кроме поведения по умолчанию окон верхнего уровня, она управляет переходом фокуса ввода между элементами управления диалогового окна.

Windows также поддерживает множество классов окон. Они реализуют такие элементы управления как кнопки, поля ввода и другие. Такие классы называются ***системными*** (предопределенными) классами.

Приложение может использовать любой существующий класс (системный или определенный приложением) для порождения нового класса. Для этого используются механизмы, называемые разбиением на субклассы (подклассы) или объединение в суперклассы.

Замечание. Следует только заметить, что:

- Приложение не должно пытаться применить операцию разбиения на субклассы или объединения в суперклассы к окну, принадлежащему другому процессу.
- В Win32 изменения в системных классах влияют только на окна этого класса внутри одного приложения. Эти изменения никак не влияют на окна в других приложениях, потому что приложения Win32 запускаются в разных адресных пространствах; таким образом, они защищены друг от друга.

Разбиение на субклассы - замещение оконной процедуры

Разбиение на суперклассы (subclassing) подразумевает замещение оконной процедуры класса окна другой процедурой. Это выполняется с помощью вызова функции SetWindowLong или SetClassLong.

Замечания:

- вызов SetWindowLong со значением индекса GWL_WNDPROC замещает оконную процедуру для определенного окна.
- в противоположность этому, вызов SetClassLong со значением индекса GCL_WNDPROC замещает оконную процедуру для всех окон этого класса, которые будут созданы после вызова SetClassLong.

Пример, в котором системный класс “button” разбивается на субклассы, обеспечивая замещение оконной процедуры. Замещающая процедура реализует специальную реакцию на получение сообщения WM_LBUTTONUP.

```
        . . .
        // глоб. переменная - для хранения адреса старой оконной функции
        FARPROC OldWndProc;
        // объявление новой оконной функции
        LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
        . . .

        . . .
        // <--- начало фрагмента кода некоторой функции ---
        // создание субкласса класса "button"
        HWND hWnd = CreateWindow("button","subclass",WS_VISIBLE|BS_CENTER,
                        100,100,200,50,NULL,NULL,hInstance,NULL);
        OldWndProc = (FARPROC)SetWindowLong(hWnd,GWL_WNDPROC,(LONG)WndProc);
        // --- конец фрагмента кода некоторой функции --->
        . . .

        // определение новой оконной функции
        LRESULT CALLBACK WndProc(HWND hWnd,UINT msg, WPARAM wParam,LPARAM
lParam)
        {
                switch(msg)
                {
                        case WM_LBUTTONUP: MessageBeep(-1); break;
                        default: return
CallWindowProc(OldWndProc,hWnd,msg,wParam,lParam);
                }
                return 0;
        }
```

Замечание. В приведенном примере разбиение на субклассы выполняется с помощью функции SetWindowLong, подразумевая, что это повлияет только на отдельное окно кнопки, для которого вызывается SetWindowLong. Если вместо этого вызвать SetClassLong, то изменится поведение всех кнопок, созданных приложением впоследствии.

Объединение в суперклассы - использование информации о классе

Объединение в суперклассы предполагает создание нового класса на основе поведения существующего. Чтобы применить эту операцию к существующему классу, приложение может использовать функцию GetClassInfo для получения описывающей этот класс структуры WNDCLASS. После соответствующего изменения этой структуры, ее можно использовать в вызове функции RegisterClass для регистрации нового класса.

Пример, в котором создается новый класс “beepbutton”, функциональные возможности которого основаны на классе по умолчанию “button”. Поведение окон стандартного класса "button" при этом не изменяется.

```
. . .
// глоб. переменная - для хранения адреса старой оконной функции
FARPROC OldWndProc;
// объявление новой оконной функции
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);
. . .
. . .
// <--- начало фрагмента кода некоторой функции ---
//  создание суперкласса "button"
WNDCLASS wc;
GetClassInfo(hInstance,"button",&wc);
wc.lpszClassName = "beepbutton";
wc.hInstance = hInstance;
OldWndProc = (FARPROC)wc.lpfnWndProc;
wc.lpfnWndProc = (WNDPROC)WndProc;
RegisterClass(&wc);

HWND hWnd = CreateWindow("beepbutton","superclassing",WS_VISIBLE |
BS_CENTER, 100, 100, 200, 50, NULL, NULL, hInstance, NULL);
// --- конец фрагмента кода некоторой функции --->
. . .

LRESULT CALLBACK WndProc(HWND hWnd,UINT msg,WPARAM wParam,LPARAM lParam)
{
switch(msg)
{
        case WM_LBUTTONDOWN: MessageBeep(-1);
default: return CallWindowProc(OldWndProc,hWnd,msg,wParam,lParam);
        }
        return 0;
}
```

Вывод: разница между двумя между механизмами разбиением на субклассы и объединением в суперклассы очевидна с точки зрения их реализации. Осталось выяснить их разницу в плане применения, определить, когда необходимо использовать разбиение на субклассы, а когда – объединение в суперклассы:

- Разбиение на субклассы изменяет поведение существующего класса.
- Объединение в суперклассы создает новый класс на основе поведения существующего класса, т.е.:
- При использовании разбиения на субклассы подразумевается изменение каждого окна каждого окна приложения, принадлежащего этому классу.

- В противоположность этому объединение в суперклассы действует только на окна, основанные на новом классе; это никак не действует на окна, основанные на исходном классе.

Контрольные вопросы

1. Что определяет (характеризует) стиль класса окна? Что определяет стиль окна?
2. Как задается стиль класса окна?
3. Для чего задаются стили класса CS_HREDRAW, CS_VREDRAW?
4. Для чего используется стиль класса CS_DBLCLKS?
5. Какая характеристика используется для уточнения внешнего вида и поведения окна, создаваемого на базе какого-либо класса?
6. Какой внешний вид и поведение обычно характерен для перекрывающихся окон (назвать стиль класса)?
7. Что такое окно-владелец и подчиненное окно? Чем поведение подчиненного окна отличается от поведения обычного перекрывающегося окна?
8. Для чего чаще всего используются временные (всплывающие, popup) окна?
9. Какой внешний вид обычно имеют временные окна (назвать стиль класса)?
10. Где располагается начало системы координат для перекрывающихся и временных окон?
11. Для чего обычно используются дочерние окна?
12. Где располагается начало системы координат для дочерних окон?
13. Какой внешний вид обычно имеют дочерние окна (назвать стиль класса)?
14. Каковы особенности поведения дочерних окон?
15. Каким простым способом можно сообщить Windows о том, что окно должно иметь горизонтальную и/или вертикальную полосу прокрутки? В чем недостатки такого метода?
16. Какое значение имеют понятия диапазон полосы прокрутки и ее текущее положение? Как можно изменять эти характеристики?
17. Какова сфера ответственности Windows в организации работы пользователя с полосой просмотра?
18. Какова сфера ответственности приложения в организации работы пользователя с полосой просмотра?
19. Какие сообщения приходят окну от полос просмотра?
20. Организован ли на системном уровне интерфейс клавиатуры для работы с полосами прокрутки?
21. При помощи какой функции можно определить системные метрики Windows?
22. Когда окно получает сообщение WM_SIZE и какие дополнительные параметры передаются окну вместе с этим сообщением?
23. Для чего предназначена функция GetClientRect?
24. Когда окно получает сообщение WM_MOVE и какие дополнительные параметры передаются окну вместе с этим сообщением?
25. Для чего предназначена функция GetWindowRect?
26. Как определить метрики шрифта, установленного в контексте отображения окна?

Задание: ответить на контрольные вопросы и прислать их на проверку (файл должен быть в формате pdf)