Программирование приложений Windows

Лекция 6. Принципы работы с элементами управления

Введение

Любое стандартное приложение Windows использует различные элементы управления, такие, как кнопки, полосы просмотра, редакторы текстов и т.д, реализованные в виде **дочерних окон**.

Дочерние окна управления (улементы управления)

- Так как дочерние окна располагаются на поверхности окна родителя, “прилипая” к ним, **приложение может создать в любом своем окне несколько элементов управления, которые будут перемещаться вместе с окном-родителем**.
- Для этого достаточно создать нужные дочерние окна, указав их размеры, расположение и некоторые другие атрибуты. После этого **приложение может взаимодействовать с элементами управления, передавая или получая от них различные сообщения**.
- Каждое **дочернее окно создается с помощью вызова функции CreateWindow.** Оконная процедура родительского окна **посылает сообщения дочерним окнам** управления, а дочерние окна управления **посылают сообщения обратно оконной процедуре**.
- Дочернее окно управления **обрабатывает сообщения мыши и клавиатуры и извещает родительское окно о том, что состояние дочернего окна изменилось**. В этом случае
- **Дочернее окно становится для родительского окна устройством ввода**. Оно инкапсулирует особые действия, связанные с графическим представлением окна на экране, реакцией на пользовательский ввод, и извещения другого окна при вводе важной информации.

Введение

Стандартные дочерние окна управления имеют вид кнопок, флажков, окон редактирования, списков, комбинированных списков, строк текста и полос прокрутки. Приложение нет необходимости беспокоиться о логике обработки мыши этими окнами, или о логике их отрисовки. Все это делается в Windows, а **все, что остается приложению – это обрабатывать сообщение WM_COMMAND**, которыми дочерние окна информируют оконную процедуру о различных событиях.

- Для **создания дочернего окна управления собственного класса**, во-первых, следует **определить класс окна и зарегистрировать** его в Windows функцией **RegisterClass**. Затем с помощью функции **CreateWindow** необходимо создать окно на основе этого класса.

Однако, Если **создается одно из предопределенных дочерних окон управления, то для этого дочернего окна класс окна регистрировать не надо**. Такой класс уже существует в Windows и имеет одно из следующих имен: “button”, ”edit”, “static”, “listbox”, “combobox” и “scrollbar”. Приложение просто использует одно из этих имен в качестве параметра в функции **CreateWindow**.

- Параметр стиля окна функции **CreateWindow** более точно определяет **вид и свойства дочернего окна управления**. Windows включает в себя оконные процедуры, обрабатывающие сообщения тех дочерних окон, которые созданы на основе перечисленных классов.

Создание дочерних окон

Вызывая **специальные функции**, можно

- **динамически перемещать** элемент управления (**MoveWindow -** функция определяет новое расположение и размеры окна в системе координат, связанной с родительским окном),
- **делать его активным или неактивным** (**EnableWindow**, для определения, является ли окно заблокированным, используется функция **IsWindowEnabled**),
- **скрывать его или отображать** (**ShowWindow**),
- **изменять заголовок** окна (**SetWindowText**),
- **искать окна (FindWindow)**
- **работать с областью окна GetClientRect**
- **уничтожить** созданный элемент управления (**DestroyWindow**).

Создание дочерних окон

- **Для создания дочернего окна элементов управления необходимо вызвать функцию CreateWindow**. С помощью этой функции создаются все окна в приложениях. Прототип функции:
- HWND CreateWindow(LPCSTR lpClassName, LPCSTR lpWindowName, DWORD dwStyle, int x, int y, int nWidth, int nHeight,HWND hWndParent, HMENU hMenu, HANDLE hInstance, LPVOID *lpParam); Параметр функции lpClassName - **указатель на строку, содержащую имя класса, на базе которого создается окно**. Для создания кнопки, например, необходимо указать имя класса “button”.
- Параметр lpWindowName - **указатель на строку, содержащую текст окна**. Для элементов управления - это либо надпись на кнопке, либо текст в поле редактирования, либо что-то иное в зависимости от типа элемента управления.
- Параметр dwStyle - **стиль создаваемого окна**. Этот параметр задается как логическая комбинация отдельных битов. **Для дочерних окон управления следует задать стиль как комбинацию констант WS_CHILD, WS_VISIBLE, WS_CLIPSIBLINGS и констант, определяющих стили элемента управления**.
- Параметры x и y определяют **положение верхнего левого угла** дочернего окна **относительно верхнего левого угла рабочей области родительского окна.**
- Параметры nWidth и nHeight определяют **ширину и высоту** создаваемого элемента управления.

Создание дочерних окон

- Параметр hWndParent определяет **дескриптор родительского окна**, на поверхности которого создается дочернее окно.
- Параметр hMenu - **идентификатор порожденного (child) окна**. Для каждого создаваемого дочернего окна **необходимо определить собственный уникальный идентификатор.** Родительское окно будет получать от дочерних окон сообщения **WM_COMMAND**. При помощи идентификатора элемента управления **оконная процедура родительского окна сможет определить дочернее окно, пославшее ей сообщение**.
- Параметр hInstance - **дескриптор экземпляра копии приложения**, которое создает окно. Необходимо использовать значение, передаваемое функции **WinMain** через параметр hInstance.
- Последний параметр lpParam используется для передачи окну **дополнительных данных**. При создании дочерних окон управления **он должен быть равен NULL**.
- Константы, **идентифицирующие стили предопределенных классов элементов управления**, определены в заголовочных файлах Windows и имеют соответственно следующие префиксы: **BS_** - “button”, **ES_** - ”edit”, **SS_** - “static”, **LBS_** - “listbox”, **CBS_** - “combobox” и **SBS_** - “scrollbar”.

Создание дочерних окон

Пример создания обычной кнопки в оконной процедуре главного окна приложнения:

```
  static UINT ID_button=1;     static HWND hWndButton;  . . .
hWndButton=CreateWindow("button","Exit",
WS_CHILD|WS_VISIBLE|WS_CLIPSIBLINGS|BS_PUSHBUTTON,
x,y,width,height,hWnd,(HMENU)ID_button,hInst,NULL);  .
```

После вызова CreateWindow оконные процедуры дочерних окон управления внутри Windows поддерживают их работу и управляют всеми процессами перерисовки (за исключением кнопки стиля **BS_OWNERDRAW**).

При необходимости созданные дочерние окна можно удалить функцией DestroyWindow. При завершении работы программы, **когда удаляется родительское окно, Windows сама удаляет и дочерние окна.**

Сообщения дочерних окон родительскому окну

Если элемент управления изменяет свое состояние, то функция родительского окна получает сообщение WM_COMMAND. Вместе с сообщением оконная процедура получает и дополнительную информацию. Эти **дополнительные параметры** имеют следующий смысл:

LOWORD(wParam) - идентификатор дочернего окна (тип UINT); HIWORD(wParam) - код уведомления (тип UINT); lParam - дескриптор дочернего окна (тип HWND). **Идентификатор дочернего окна** – это значение, **передаваемое функции CreateWindow,** когда создается рабочее окно.

Дескриптор дочернего окна – это значение, которое Windows **возвращает при вызове функции CreateWindow**.

Код уведомления – это дополнительный код, который дочернее окно использует для того, чтобы **сообщить родительскому окну более точные сведения о сообщении**.

Константы, **идентифицирующие различные коды уведомления**, определены в заголовочных файлах Windows и имеют соответственно следующие префиксы: **BN_** - “button”, **EN_** - “edit”, **LBN_** - “listbox”, **CBN_** - “combobox” и **SB_** - “scrollbar”.

Сообщения дочерних окон родительскому окну

Пример **оконной процедуры родительского окна, в котором происходит обработка нажатия на дочернее окно-кнопку** с идентификатором ID_button:

```
case WM_COMMAND: {

  UINT idCtl=LOWORD(wParam); // идентификатор дочернего окна

  UINT code=HIWORD(wParam);  // код уведомления

  HWND hChild=(HWND)lParam;  // дескриптор дочернего окна

  if(idCtl==ID_button&&code==BN_CLICKED)  { // кнопка была нажата

        CloseWindow(hWnd); // закрыть окно-родителя
  }
}; return 0;
```

Сообщения родительского окна дочерним окнам

Родительское окно также может передавать сообщения дочерним окнам, в ответ на которые это дочернее окно будет выполнять различные действия. Для этого необходимо знать **дескриптор дочернего окна**.

Передать можно как обычное оконное сообщение (с префиксом **WM_**), так и специфические для каждого типа элемента управления

Константы, **идентифицирующие различные сообщения для дочерних окон управления**, определены в заголовочных файлах Windows и имеют соответственно следующие префиксы: **BM_** - “button”, **EM_** - “edit”, **LB_** - “listbox”, **CB_** - “combobox”. Для работы с окнами класса “scrollbar” применяются специальные **Set/Get**-функции WinAPI.

Существует два способа передачи сообщений.

- **Первый способ передачи сообщений - запись сообщения в очередь приложения**. Он основан на использовании функции **PostMessage**. **Записанное при помощи функции PostMessage сообщение будет выбрано и обработано в цикле обработки сообщений**.

  (Асинхронные сообщения)
- **Второй способ передачи сообщений - непосредственная передача сообщения функции окна, минуя очередь сообщений**. Этот метод реализуется функцией **SendMessage**. Функция **SendMessage** вызывает **функцию окна и возвращает управление только после возврата из функции окна,** которому передано сообщение.

  (Синхронные сообщения)

Сообщения родительского окна дочерним окнам

Каждое дочернее окно имеет дескриптор окна (тип HWND) и идентификатор окна (тип UINT), которые являются уникальными среди других. Знание одного из этих элементов позволяет приложению получить другой.

Если **известен дескриптор** hWndChild дочернего окна, то **можно получить его идентификатор**:

```
UINT id=GetWindowLong(hWndChild,GWL_ID);
```

Также можно использовать **другую функцию** (хотя часть “Dlg” имени функции относится к окну диалога, на самом деле это функция общего назначения):

```
UINT id=GetDlgCtrlID(hWndChild);
```

Зная идентификтор id дочернего окна, **можно получить его дескриптор**:

```
HWND hWndChild=GetDlgItem(hWndParent,id);
```

Пример сообщения родительского окна hWnd дочернему окну класса “edit”.

Окну редактирования с дескриптором hWndEdit передается сообщение об установке максимального количества вводимых символов (5 символов):

```
static UINT ID_edit=3;
static HWND hWndEdit;      . . .
hWndEdit=CreateWindow("edit",NULL, WS_CHILD|WS_VISIBLE|WS_CLIPSIBLINGS|
WS_BORDER|ES_LEFT, x,y,width,height,hWnd,(HMENU)ID_edit,hInst,NULL);
. . .
SendMessage(hWndEdit,EM_LIMITTEXT,5,0L);
```

Дочерние окна и фокус ввода

- **Дочерние элементы управления получают фокус при щелчке мыши на них**. Когда дочерние окна управления **получают фокус ввода**, родительское окно **теряет его**; весь **ввод с клавиатуры направляется теперь не на родительское окно, а на дочернее окно управления**.
- Такая ситуация создает очевидную проблему: **приложение теряет контроль над обработкой сообщений от клавиатуры**.
- Примером является то, что перекрывающееся окно не имеет возможности обеспечить переход пользователя от элемента управления к элементу управления с помощью клавиш <Tab> и <Shift+Tab>.
- **Решение этой проблемы состоит в использовании приема, называемого созданием подкласса окна** (window subclassing, разбиение на подклассы, установка новой оконной процедуры).
- Оконная процедура дочерних окон управления находится в недрах Windows. Однако, можно **получить адрес этой оконной процедуры** с помощью вызова **GetWindowLong**, в котором в качестве параметра используется идентификатор **GWL_WNDPROC**.

Дочерние окна и фокус ввода

- **Вызывая функцию SetWindowLong можно не только получить адрес оконной процедуры дочернего окна управления, но и установить новую оконную процедуру**.
- Это очень мощный прием, **он позволяет** “влезть” в существующие внутри Windows оконные процедуры, **обработать некоторые сообщения** специфическим для приложения способом**, а все остальные сообщения оставить прежней оконной процедуре**.
- Так, например, новые оконные процедуры дочерних окон управления при получении сообщений о нажатии клавиш <Tab> и <Shift+Tab> могут просто передать фокус ввода функцией **SetFocus** следующему (или предыдущему) дочернему окну управления. Для обработки же остальных сообщений они могут вызвать старые оконные процедуры соответствующих окон управления с помощью функции **CallWindowProc**.

Дочерние окна управления и цвет

Чтобы дочерние окна управления выглядели привлекательно, необходимо **либо как-то изменить цвет рабочей области окна**, согласовывая его с цветами окон управления**;**

- **Изменить настройку цветов окон управления**.
- **Изменение цвета рабочей области окна**

Рассмотрим теперь **первый способ решения проблемы цвета**. Предварительно обсудим, как Windows использует системные цвета (system colors).

Windows поддерживает 25 системных цветов, предназначенных для отрисовки различных элементов экрана. Приложение может получить или установить текущие значения этих цветов (на время текущего сеанса Windows) с помощью функций **GetSysColor** и **SetSysColor** и идентификаторов системных цветов. **Идентификаторы системных цветов** с префиксом **COLOR_,** определены в заголовочных файлах Windows.

Например, цвет **COLOR_BTNFACE** является основным цветом поверхности нажимаемых кнопок и цветом фона остальных (этот же цвет используется в окнах диалога и сообщений). А цвет **COLOR_WINDOWTEXT** для флажков и радио-переключателей используется как цвет текста.

- Теперь **приведем цвет фона рабочей области родительского окна, цвет фона текста и цвет самого текста в соответствие с, например, цветами кнопок**. Сначала обсудим, как можно изменить цвет рабочей области родительского окна.

Дочерние окна управления и цвет

- **При определении класса родительского окна для фона рабочей области можно использовать системный цвет**, например **COLOR_BTNFACE**:
- wndclass.hbrBackground=(HBRUSH)(COLOR_BTNFACE+1); Windows понимает, что если значение поля hbrBackground такое низкое, то оно фактически ссылается на системный цвет, а не реальный дескриптор кисти. Windows в этом случае требует, чтобы **при использовании идентификаторов системных цветов к ним прибавляли 1** (чтобы это значение не стало равным NULL).
- Если выводить текст на экран, используя функцию **TextOut**, то Windows для цвета фона текста (цвет, который стирает фон позади текста) и цвета самого текста использует значения, определенные в контексте устройства. По умолчанию это белый (фон) и черный (текст) цвета вне зависимости от системных цветов и поля hbrBackground структуры класса окна.

Дочерние окна управления и цвет

- **Для изменения цвета и фона текста, выводимого в рабочей области родительского окна, на системные цвета**, необходимо после получения дескриптора hDC контекста устройства для окна родительского окна вызвать функции:
- SetBkColor(hDC,GetSysColor(COLOR_BTNFACE));
SetTextColor(hDC,GetSysColor(COLOR_WINDOWTEXT));
- В итоге цвет фона рабочей области родительского окна, цвет фона текста и цвет самого текста приведены в соответствие с цветом кнопок.

Настройка цветов дочерних окон управления

- Приступим к обсуждению **решения проблемы цвета вторым способом**.
- **Когда дочернее окно управления собирается рисовать свою рабочую область, оно посылает процедуре родительского окна соответствующее сообщение WM_CTLCOLOR…** (например, кнопка посылает сообщение **WM_CTLCOLORBTN**).
- **Родительское окно на основании этого сообщения может менять цвета,** которые будет использовать оконная процедура дочернего окна при рисовании.
- В итоге
- **Родительское окно может управлять цветами своих дочерних окон**. Именно обработка сообщений запроса дочерними окнами информации о цвете и есть второй способ решение проблемы несогласованности цветов.

Настройка цветов дочерних окон управления

- Рассмотрим пример управления цветом кнопки. Когда кнопка собирается рисовать свою рабочую область, она посылает процедуре родительского окна сообщение **WM_CTLCOLORBTN**. Родительское окно на основании этого сообщения может менять цвета, которые будет использовать оконная процедура кнопки при рисовании.
- **Когда оконная процедура родительского окна получает сообщение WM_CTLCOLORBTN**, то параметр wParam этого сообщения является дескриптором контекста устройства кнопки, а параметр lParam – дескриптором окна кнопки**.**
- **Родительское окно в случае обработки этого сообщения в своей оконной процедуре**: необязательно устанавливает цвет фона текста функций **SetBkColor;** необязательно устанавливает цвет текста функций **SetTextColor;** обязательно возвращает дескриптор кисти дочернему окну.
- **Теоретически, дочернее окно использует кисть для рисования фона.** Необходимо удалить кисть, когда она становится ненужной.

Настройка цветов дочерних окон управления

- Приведем **пример обработки сообщения WM_CTLCOLORBTN родительским окном**. Предполагается, что кисть hBrushBack создается оконной процедурой родительского окна hWnd при обработке сообщения **WM_CREATE** и удаляется при обработке сообщения **WM_DESTROY**:
- case WM_CTLCOLORBTN:

```
{   HDC hdcButton = (HDC) wParam;
HWND hChild = (HWND) lParam;          SetBkColor(hdcButton,RGB(0,255,0));
    SetBkMode(hdcButton,TRANSPARENT);        SetTextColor(hdcButton, RGB(0,0,0));

POINT point; point.x=point.y=0;
ClientToScreen(hWnd,&point);          SetBrushOrg(hdcButton,point.x,point.y);
      UnrealizeObject(hBrushBack);
}; return(hBrushBack);
```

Кнопки

- **Для создания кнопки приложение должно создать дочернее окно на базе предопределенного класса “button”**, например**:**
- static UINT ID_button=1; static HWND hWndButton; . . . hWndButton=CreateWindow**("button"**,"Text", WS_CHILD|WS_VISIBLE|WS_CLIPSIBLINGS|**BS_PUSHBUTTON**, x,y,width,height, **hWndParent**,(HMENU)**ID_button**,hInst,NULL); **Родительское окно будет получать от кнопки сообщение WM_COMMAND с кодом уведомления BN_CLICKED**. Этим сообщением кнопка информирует родительское окно о том, что с ней что-то сделали. **Для обработки сообщения оконная функция родительского окна может содержать код следующего вида**:
- case WM_COMMAND: { UINT idCtl=LOWORD(wParam); // идентификатор дочер. окна UINT code=HIWORD(wParam); // код уведомления HWND hChild=(HWND)lParam; // дескриптор дочер. окна **if(idCtrl==ID_button&&code==BN_CLICKED)** { // сообщение о том, что нажата кнопка // с идентификатором ID_button . . . } }; return 0;

Кнопки

- **Родительские окна могут посылать следующие сообщения кнопкам**:
- **BM_GETCHECK и BM_SETCHECK** - для установки и снятия меток типа “включено/выключено” флажков-переключателей и радио-переключателей;
- **BM_GETSTATE и BM_SETSTATE** - для установки состояния “нажата/отпущена” всех типов кнопок;
- **BM_SETSTYLE** - для изменения стиля любой кнопки после ее создания.
- Следует заметить, что
- **Только нажимаемые кнопки и кнопки, определяемые пользователем, посылают своему родительскому окну сообщение WM_CTLCOLORBTN**.
- Кроме того, только **кнопки, определяемые пользователем**, реагируют на обработку сообщения родительским окном, **используя кисть** для закрашивания фона.
- А это совершенно бесполезно, поскольку за рисование кнопок, определяемых пользователем, и так отвечает родительское окно.

Нажимаемые кнопки

- **Нажимаемые кнопки** (push buttons) представляют собой прямоугольник, внутри которого находится текст, заданный в параметре текста окна функции **CreateWindow**.
- Нажимаемые кнопки управления используются в основном **для запуска немедленного действия** без сохранения какой-либо индикации кнопки типа “включено/выключено”. Эти два типа нажимаемых кнопок управления имеют стили, которые называются **BS_PUSHBUTTON** и **BS_DEFPUSHBUTTON** (символы “DEF” означают “по умолчанию – default”).
- Функционирование кнопок этих двух стилей **при использовании их в диалоговых окнах отличается друг от друга.**
- Если же их **использовать в обычных перекрывающихся окнах, то эти два типа нажимаемых кнопок действуют одинаково**, хотя кнопка **BS_DEFPUSHBUTTON** имеет более жирную рамку.
- Когда курсор мыши находится на нажимаемой кнопке и левая клавиша мыши нажата, то кнопка перерисовывается так, чтобы выглядеть нажатой.
- **Отпускание клавиши мыши**, когда курсор мыши находится на нажимаемой кнопке, приводит **к восстановлению облика кнопки и посылке родительскому окну сообщения WM_COMMAND** с кодом нотификации **BN_CLICKED**.

Нажимаемые кнопки

- Приложение может **имитировать нажатие кнопки**, посылая окну сообщение **BM_SETSTATE**. Следующий оператор приводит к “нажатию” кнопки:
- SendMessage(hWndButton,BM_SETSTATE,1,0l); // wParam=1 – нажата
  Следующий вызов заставляет кнопку **вернуться к своему нормальному состоянию**:
- SendMessage(hWndButton,BM_SETSTATE,0,0l); // wParam=0 – отпущена
  Также можно послать нажимаемой кнопке сообщение BM_GETSTATE. Дочерняя кнопка управления возвращает текущее состояние – TRUE, если кнопка нажата и FALSE (или 0), если она в обычном состоянии:
- int press= SendMessage(hWndButton,BM_GETSTATE,0,0l); if(press) { /* кнопка нажата */ } else { /* кнопка в нормальном состоянии */ } **Замечание.** Поскольку **нажимаемая кнопка не сохраняет информацию о своем положении типа “включено/выключено”,** сообщения **BM_GETCHECK** и **BM_SETCHECK** не используются.

Флажки-переключатели

- **Флажки** (check boxes) представляют собой маленькие квадратные окна с размещенным обычно справа от окна текстом (если при создании кнопки используется стиль **BS_LEFTTEXT**, то текст окажется слева).
- **Флажки, как правило, действуют как двухпозиционные переключатели:** один щелчок вызывает появление контрольной метки (состояние “включено”); другой щелчок приводит к исчезновению этой метки (состояние “выключено”).
- В приложениях **флажки обычно объединяются, что дает пользователю возможность установить опции.** Двумя наиболее используемыми стилями для флажков являются **BS_CHECKBOX** и **BS_AUTOCHECKBOX.**
- При использовании стиля **BS_CHECKBOX приложение само должно устанавливать контрольную метку,** посылая сообщение **BM_SETCHECK**. В этом случае **обработка родительским окном** сообщения **WM_COMMAND** с кодом нотификации **BN_CLICKED** от флажка с идентификатором ID_button могла бы быть следующей:
- case WM_COMMAND: { UINT idCtl=LOWORD(wParam); // идентификатор дочер. окна UINT code=HIWORD(wParam); // код уведомления HWND hChild=(HWND)lParam; // дескриптор дочер. окна if(idCtrl==ID_button&&code==BN_CLICKED) { int cur_label= SendMessage(hWndButton,**BM_GETCHECK**,0,0l); SendMessage(hChild,**BM_SETCHECK**,(WPARAM)(**!cur_label**),0l); } }; return 0;

Флажки-переключатели

- При стиле **BS_AUTOCHECKBOX флажок самостоятельно включает и выключает контрольную метку,** и оконная процедура родительского окна **может игнорировать** сообщения **WM_COMMAND**.
- Если приложению необходимо **инициализировать флажок меткой** (установить состояние “включено”), то ему следует послать сообщение **BM_SETCHECK** с параметром wParam, равным 1 (значение 0 снимает с флажка метку):
- SendMessage(hWndButton,BM_SETCHECK,1,0l); Если необходимо узнать **текущее состояние флажка**, то для этого можно послать сообщение **BM_GETCHECK**:
- int iCheck=SendMessage(hWndButton,BM_GETCHECK,0,0l); Полученное значение равно TRUE (не равно 0), если флажок отмечен (“включен”) или FALSE (или 0), если не отмечен.
- **Двумя другими стилями флажков являются BS_3STATE и BS_AUTO3STATE**. Как показывают их имена, **эти стили могут отображать третье состояние – серый цвет внутри окна флажка** – которое имеет место, когда такому флажку посылается сообщение **BM_SETCHECK** с параметром, равным 2.
- Серый цвет **показывает пользователю, что его выбор неопределен или не имеет отношения к делу.** В этом случае флажок **не может быть включен** – т.е. он запрещает какой-либо выбор в данный момент. Однако **флажок продолжает посылать сообщения** родительскому окну, если щелкать на нем мышью.

Радио-переключатели

- **Радио-переключатели** (radio buttons, радио-кнопки) похожи на флажки, но их форма не квадратная, а круглая. Жирная точка внутри флажка показывает, что переключатель отмечен.
- **Радио-кнопка может иметь стиль окна BS_RADIOBUTTON или BS_AUTORADIOBUTTON**, но последний используется только в окнах диалога. В окнах диалога группы радио-переключателей, как правило**, используются для индикации нескольких взаимоисключающих опций.** В отличие от флажков, если повторно щелкнуть на радио-кнопке, то ее состояние не изменится.
- **При получении сообщения WM_COMMAND с кодом нотификации BN_CLICKED** от радио-переключателя с идентификатором ID_button, **необходимо отобразить его отметку, отправив сообщение BM_SETCHECK** с параметром wParam, равным 1**. Для остальных переключателей этой группы можно отключить контрольную метку**, послав сообщение **BM_SETCHECK** с параметром wParam, равным 0:
- case WM_COMMAND: { UINT idCtl=LOWORD(wParam); // идентификатор дочер. окна UINT code=HIWORD(wParam); // код уведомления HWND hChild=(HWND)lParam; // дескриптор дочер. окна if(idCtrl==ID_button&&code==BN_CLICKED) { SendMessage(hChild,BM_SETCHECK,1,0l); // для всех остальных радиопереключателей группы - снять метки SendMessage(hWndOtherButton,BM_SETCHECK,0,0l); . . . } }; return 0;

Окна групп

- **Окна групп** (group boxes) – стиль **BS_GROUPBOX** – является исключением в классе кнопок. Они **не обрабатывает ни сообщения от клавиатуры, ни сообщения от мыши, оно не посылает родительскому окну сообщений WM_COMMAND.**
- Окно группы представляет собой прямоугольную рамку с текстом наверху. Окна групп часто **используют для того, чтобы в них размещать другие дочерние окна управления**.

Кнопки, определяемые пользователем

- **Если есть необходимость полного управления внешним видом кнопки, но нет желания реализовывать логику обработки клавиатуры и мыши**, то можно создать кнопку стиля **BS_OWNERDRAW.**
- Большинство программ, в которых для рисования собственных кнопок используется стиль кнопки **BS_OWNERDRAW**, часто для изображения применяют небольшие растровые образы.
- Кнопки стиля **BS_OWNERDRAW при необходимости перерисовки посылают своему родительскому окну сообщение WM_DRAWITEM**. Это **происходит**
- при первоначальном **создании** кнопки,
- при ее **нажатии** или **отпускании**,
- при получении или потере **фокуса** ввода
- и во всех других случаях, когда требуется **перерисовка**.
- **При обработке сообщения WM_DRAWITEM** параметр lParam этого сообщения является указателем на структуру типа **DRAWITEMSTRUCT**:
- case WM_DRAWITEM: { UINT idCtl = (UINT) wParam; DRAWITEMSTRUCT *lpdis = (LPDRAWITEMSTRUCT) lParam; // необходимо перерисовать окно с идентификатором idCtl . . . }; return 0l;

Кнопки, определяемые пользователем

- **Структура DRAWITEMSTRUCT содержит информацию, необходимую программе для рисования кнопки** (такая же структура используется для создания определяемых пользователем списков). Полями структуры, которые важны для работы с кнопками, являются
- hDC - **контекст устройства** для кнопки,
- rcItem - **структура RECT** с размерами кнопки,
- CtlID - **идентификатор окна** управления,
- itemState - показывает, **нажата ли кнопка и имеет ли она фокус ввода**. Если кнопка в момент сообщения нажимается, то в поле itemState устанавливается бит. **Для выяснения, установлен ли бит**, можно использовать константу **ODS_SELECTED**. Если кнопка имеет фокус ввода, тогда будет установлен бит **ODS_FOCUS** поля itemState.
- **Замечание.** Необходимо заметить, что **для отрисовки дочерней кнопки, определяемой пользователем, Windows сама получает контекст устройства кнопки** и включает его в виде поля в структуру, передаваемую вместе с сообщением **WM_DRAWITEM**.
- После использования этого контекста устройства **программа должна оставить его в том же состоянии, в каком он ей передавался**. Все объекты GDI, выбранные в контекст устройства, должны быть из него удалены.

Статическое дочернее окно управления

- **Статическое дочернее окно управления** - это окно, создаваемое на базе предопределенного класса ”static”.
- Статические окна нельзя использовать для управления работой приложения: **они не воспринимают щелчки мыши и не обрабатывают сообщения от клавиатуры, не посылает родительскому окну сообщение WM_COMMAND**. Обычно этот орган управления **используется для оформления внешнего вида диалоговых панелей или окон приложения**.
- **Все сообщения от мыши через “прозрачное” окно статического дочернего окна попадают в родительское окно**.
- Для **создания статического дочернего окна** необходимо использовать функцию **CreateWindow**. В качестве первого параметра **следует указать класс окна “static**”, например:
- static UINT ID_static=2; static HWND hWndStatic; . . .
  hWndStatic=CreateWindow(**"static"**,NULL,
  WS_CHILD|WS_VISIBLE|WS_CLIPSIBLINGS| **SS_GRAYRECT**, x,y,width,height,
  hWndParent,(HMENU)**ID_static**,hInst,NULL);

Статическое дочернее окно управления

- **Рассмотрим стили статического дочернего окна**
- Первые шесть стилей – **SS_BLACKRECT, SS_GRAYRECT, SS_WHITERECT** и **SS_BLACKFRAME, SS_GRAYFRAME, SS_WHITEFRAME** – рисуют **закрашенный определенным цветом прямоугольник или прямоугольную рамку, нарисованную линией определенного цвета без закрашивания внутренней области**. Поле текста окна функции **CreateWindow** для этих стилей игнорируется.
- Вставки “BLACK”, “GRAY”, “WHITE” не означают, что цветами являются соответственно черный, серый и белый. Эти цвета основаны на системных цветах: “BLACK” – **COLOR_3DDKSHADOW**, “GRAY” – **COLOR_BTNSHADOW**, “WHITE” – **COLOR_BTNHIGHLIGHT.**
- Для создания **рамки с тенью, состоящей из серого и белого цветов** можно дополнительно воспользоваться стилями **SS_ETCHEDHORZ, SS_ETCHEDVERT, SS_ETCHEDFRAME.**
- Статический класс окон также включает в себя три стиля текста **– SS_LEFT, SS_RIGHT и SS_CENTER**. Они предназначены **для выравнивания текста соответственно по левому краю, правому краю и центру.** Текст задается в параметре текста окна функции **CreateWindow**, и позднее может быть изменен функцией **SetWindowText**.
- Фоном дочерних окон этих трех стилей обычно является **COLOR_BTNFACE**, а самого текста – **COLOR_WINDOWTEXT**.
- **Замечание.** Кроме перечисленных стилей, статический класс содержит стили окна **SS_ICON** и **SS_USERITEM**. Однако статические окна таких стилей **имеют смысл только при использовании их в окнах диалога**.

Поля ввода

- В Windows зарегистрирован класс окна с именем “edit”, на базе которого можно создать **однострочный или многострочный текстовый редактор**.
- Для того чтобы **создать свой собственный редактор, достаточно создать на базе класса “edit”** окно управления, вызвав функцию **CreateWindow**, например:
- static UINT ID_edit=3; static HWND hWndEdit; . . . hWndEdit=CreateWindow**("edit"**,NULL, WS_CHILD|WS_VISIBLE|WS_CLIPSIBLINGS| **WS_BORDER|ES_LEFT**, x,y,width,height,hWnd,(HMENU)**ID_edit**,hInst,NULL);
- **По умолчанию в окне редактирования отсутствует рамка окна.** Добавить ее можно, используя стиль **WS_BORDER**.
- Для того чтобы после создания **поместить в редактируемое поле текст**, можно воспользоваться функцией **SetWindowText.** Для **получения текста из окна редактирования** используются функии **GetWindowTextLength** и **GetWindowText**.
- Управляющие окна редактирования хранят текст в области памяти программы (в адресном пространстве программы). Содержимое управляющих полей редактирования **ограничено примерно 32 килобайтами.**

Поля ввода

- **Рассмотрим стили окон редактирования**
- **Текст в управляющих окнах редактирования может быть выравнен либо по правому краю, либо по правому, либо по центру.** Формат можно задать с помощью стилей окна **ES_LEFT, ES_RIGHT** и **ES_CENTER**.
- По умолчанию в управляющем окне редактирования имеется одна строка. Для создания такого **окна с автоматической горизонтальной прокруткой** следует добавить стиль **ES_AUTOHSCROLL.**
- Приложение может создать **многострочное управляющее окно редактирования**, используя стиль окна **ES_MULTILINE**. Для многострочного окна редактирования, если не задан стиль **ES_AUTOHSCROLL**, то текст автоматически переносится на новую строку при достижении правого края окна.
- При задании стиля **ES_AUTOHSCROLL** в многострочном редакторе для перехода на новую строку нужно нажимать клавише <Enter> (т.е. **появляется возможность горизонтальной прокрутки**). Используя стиль окна **ES_AUTOVSCROLL**, в многострочное окно редактирования можно **включить возможность вертикальной прокрутки**.
- Если стили **ES_AUTOHSCROLL** и **ES_AUTOVSCROLL** включены в многострочные управляющие окна, то **можно добавить и сами полосы прокрутки**.
- **Добавление полос прокрутки** делается путем использования тех же идентификаторов стиля окна, что и для недочерних окон: **WS_HSCROLL** и **WS_VSCROLL**.
- Чаще всего **для многострочных редакторов используется следующая комбинация стилей**:
- WS_CHILD|WS_VISIBLE|WS_CLIPSIBLINGS|WS_BORDER| ES_MULTILINE|ES_|ES_AUTOVSCROLL|WS_HSCROLL|WS_VSCROLL

Поля ввода

- **Рассмотрим, как окна редактирования уведомляют родительское окно о тех изменениях, которые с ними происходят**
- Поля редактирования **посылают оконной процедуре родительского окна сообщения WM_COMMAND**. Значения параметров сообщения wParam и lParam такие же, как и для кнопок. Ниже представлены **коды уведомления управляющих окон редактирования:**
- **EN_SETFOCUS** – окно получило фокус ввода.
- **EN_KILLFOCUS** – окно потеряло фокус ввода.
- **EN_CHANGE** – содержимое окна будет меняться.
- **EN_UPDATE** – содержимое окна изменилось.
- **EN_ERRSPACE** – произошло переполнение буфера редактирования.
- **EN_MAXTEXT** – произошло переполнение буфера редактирования при вставке.
- **EN_HSCROLL** – на горизонтальной полосе прокрутки был щелчок мышью.
- **EN_VSCROLL** – на вертикальной полосе прокрутки был щелчок мышью.
- **Для обработки сообщения от поля редактирования оконная функция родительского окна** может содержать код следующего вида:
- case WM_COMMAND: { UINT idCtl=LOWORD(wParam); // идентификатор дочер. окна UINT code=HIWORD(wParam); // код уведомления HWND hChild=(HWND)lParam; // дескриптор дочер. окна if(idCtrl==ID_edit&&code==EN_CHANGE) { // текст поля сейчас будет меняться . . . } }; return 0;

Поля ввода

- **Родительские окна также могут посылать окнам редактирования сообщения**
- Специфических сообщений, которые **родительское окно может послать окну редактирования** при помощи функции **SendMessage**, достаточно много. Рассмотрим наиболее часто употребляемые:
- **WM_CUT -** для удаления с пересылкой в буфер обмена;
- **WM_COPY -** копирования в буфер обмена;
- **WM_CLEAR -** очищения выделенной части текста из окна редактирования;
- **WM_PASTE -** для вставки текста из буфера обмена в окно редактирования;
- **EM_GETSEL -** для получения начальной и конечной позиции текущего выделения текста;
- **EM_SETSEL -** для выделения некоторого участка текста;
- **EM_REPLACESEL** - для замены текущего выделенного текста другим текстом;
- **EM_GETLINECOUNT** - для получения число строк многострочного редактора;
- **EM_LINEINDEX** - для получения смещения от начала буфера до некоторой строки;
- **EM_LINELENGTH -** для получения длины некоторой строки;
- **EM_GETLINE** - для копирования некоторой строки в буфер программы.

Элемент управления «Список»

- **С помощью класса “listbox” можно создавать одноколоночные и многоколоночные списки**, имеющие вертикальную (для одноколоночных списков) и горизонтальную (для многоколоночных списков) полосу просмотра.
- Список может быть **как с одиночным выбором, так и с множественным**. Последний позволяет пользователю выбирать более одного пункта списка
- **Для создания списка приложение должно вызвать функцию CreateWindow, передав ей в качестве первого параметра указатель на строку “listbox”**, например:
- static int ID_list=4; static HWND hWndList; . . . hWndList=CreateWindow(**"listbox",**NULL, WS_CHILD|WS_VISIBLE|WS_CLIPSIBLINGS| **LBS_STANDARD**, x,y,width,height,hWnd,(HMENU)**ID_list**,hInst,NULL); **Окно списка посылает сообщение WM_COMMAND своему родительскому окну,** когда в списке выбирается какой-либо пункт. Родительское окно **может определить, какой пункт списка выбран**.
- **Замечание.** При необходимости родительское окно может создать список, внешний вид которого определяется пользователем, и само отрисовывать изображения элементов списка.

Элемент управления «Список»

- **Рассмотрим стили окон списков**
- **При задаваемом по умолчанию стиле окна списка (**только стиль WS_CHILD) сообщения **WM_COMMAND** родительскому окну **не посылаются**. Это означает, что программе следует опрашивать окно списка посредством передачи ему сообщений относительно выбранных в списке пунктов.
- В окна списка **почти всегда** включают идентификатор стиля **LBS_NOTIFY**, **что позволяет родительскому окну получать от окна списка** сообщение **WM_COMMAND**.
- Если приложение желает иметь **возможность сортировать элементы** списка, ему необходимо использовать в окне списке и другой часто используемый идентификатор стиля – **LBS_SORT.**
- **По умолчанию, в списке допускается выбор только одного пункта.** Если необходимо создать список с **возможностью выборки сразу нескольких пунктов**, то следует использовать стиль **LBS_MULTIPLESEL**.
- **По умолчанию,** оконная процедура окна списка выводит только список элементов **без какой-либо рамки** вокруг. **Рамку можно добавить** с помощью стиля **WS_BORDER**.
- Для **прокрутки содержимого с помощью мыши и вертикальной полосы прокрутки** следует использовать стиль **WS_VSCROLL**.

Элемент управления «Список»

- В заголовочных файлах Windows определяется **стиль списка, который называется LBS_STANDARD.** Стиль **LBS_STANDARD** включает в себя **наиболее часто используемые стили списка**. Он определяется как комбинация
- (LBS_NOTIFY|LBS_SORT|WS_VSCROLL|WS_BORDER)
- **Замечание.** Также можно пользоваться стилями **WS_SIZEBOX** и **WS_CAPTION**, которые дают возможность менять **размер окна и перемещать его** по рабочей области родительского окна.
- **Рассмотрим сообщения, которые могут посылать родительские окна окнам списков для их заполнения и дальнейшей работы с ними**
- После создания окна списка в него следует добавить строки текста – элементы списка. Это делается при помощи функции **SendMessage** посредством отправки сообщений окну списка. Ссылка на строку текста (на элемент) обычно осуществляется через индекс, который начинается с 0, что соответствует самому верхнему элементу списка.

Элемент управления «Список»

- Специфических **сообщений, которые родительское окно может послать окну списка** при помощи функции **SendMessage**, достаточно много. Рассмотрим наиболее часто употребляемые:
- **LB_ADDSTRING** - для добавления строки в конец списка или по алфавитному порядку (последнее - если используется стиль **LBS_SORT**);
- **LB_INSERTSTRING -** для вставки строки на определенное порядковое место в списке (если не используется стиль **LBS_SORT**);
- **LB_DELETESTRING** - для удаления строки с некоторым порядковым номером;
- **LB_RESETCONTENT** - для полного очищения списка;
- **LB_GETCOUNT** - для определения общего количества пунтов списка;
- **LB_GETTEXTLENGTH** - для определения длины любого пункта списка;
- **LB_SETCURSEL -** для установки элемента, выбираемого по умолчания (для списков с единичной выборкой);
- **LB_SELECTSTRING -** для выборки (установки) элемента списка на основе его первых символов (для списков с единичной выборкой);
- **LB_GETCURSEL -** для определения индекса элемента, выбранного в текущий момент (для списков с единичной выборкой);
- **LB_SETSEL -** для установки состояния выбора конкретного элемента, при этом не оказывая влияния на другие элементы, которые могли бы быть выбранными (для списков с множественным выбором);
- **LB_GETSEL** - для определения, выбран ли конкретный элемент списка или нет.

Элемент управления «Список»

- **Самым мощным сообщение для окон списка является сообщение LB_DIR**. Следующий оператор заполняет список перечнем файлов каталога, иногда с подкаталогами и именами доступных дисков:
- SendMessage(hWndList,LB_DIR, iAttr,(LPARAM)szFileSpec);
- Параметр iAttr **–** это код атрибута файла, он определяет **атрибуты отображаемых в списке файлов**.
- Параметр szFileSpec – это указатель на строку, задающую **спецификацию файлов**. Такая спецификация не влияет на подкаталоги, которые содержатся в списке.
- Рассмотрим, какие значения, объединенные порязрядным ИЛИ, можно использовать **для задания атрибутов файлов**:
- В качестве младшего байта используется **обычный атрибут файла**: DDL_READWRITE – обычный файл; DDL_READONLY – файл только для чтения, DDL_HIDDEN – скрытый файл; DDL_SYSTEM – системный файл; DDL_DIRECTORY – подкаталог; DDL_ARCHIVE – файл с установленным архивным битом.
- Старший байт обеспечивает некоторый **дополнительный контроль над выводимыми именами**: DDL_DRIVES – включение имен дисков; DDL_EXCLUSIVE – включать только файлы с указаными атрибутам.

Элемент управления «Список»

- Вот как будет, например, выглядеть вызов, выводящий в список имена **всех файлов** текущего каталога, имена **всех подкаталогов** этого каталога (в виде [subdir]) и **имена всех доступных дисков** (в виде [-A-]):
- SendMessage(hWndList,LB_DIR, DDL_READWRITE|DDL_READONLY|DDL_HIDDEN| DDL_SYSTEM|DDL_DIRECTORY|DDL_DRIVES|DDL_ARCHIVE, (LPARAM)"*.*");
- Если для списка **не используется** стиль **LBS_SORT**, то имена файлов и каталогов выводятся вперемешку, а имена доступных дисков оказываются в конце списка.
- **Замечание.** Когда **пользователь щелкает мышью над окном списка**, окно списка получает фокус ввода. **Если окно списка имеет фокус ввода, то для выбора пунктов** можно пользоваться **как мышью, так и клавиатурой**.
- **Рассмотрим сообщения, которые могут посылать списки родительским окнам**
- **Окно списка посылает родительскому окну сообщения WM_COMMAND**, в старшем слове параметра wParam этого сообщения содержится код уведомления:
- **LBN_ERRSPACE** - превышен размер памяти, отведенный для списка
- **LBN_SELCHANGE** - изменен текущий выбор (подсветка перемещается по списку);
- **LBN_DBLCLK** - на пункте списка имел место двойной щелчок мышью.
- **Замечание.** Окно списка посылает коды уведомления **LBN_SELCHANGE** и **LBN_DBLCLK** только в том случае, если в стиль дочернего окна **включен идентификатор LBS_NOTIFY.**

Элемент управления «Список»

- **Для обработки сообщения от списка оконная функция родительского окна** может содержать код следующего вида:
- case WM_COMMAND:

```
{   UINT idCtl=LOWORD(wParam); // идентификатор дочер. окна      UINT
    code=HIWORD(wParam);  // код уведомления
HWND hChild=(HWND)lParam;  // дескриптор дочер. окна
    If(idCtrl==ID_listbox&&code==LB_SELCHANGE)
{           // текущий выбор в списке изменился              . . .
}
}; return 0;
```

Выпадающий список

- Этот класс является **комбинацией списка и однострочного редактора**, поэтому для комбинированного списка используются **стили, коды извещения и сообщения, аналогичные списку ”listbox”, а также некоторые сообщения, специфические для редактора текста класса “edit”.**
- Для того **чтобы создать список класса “combobox”,** приложение должно вызвать функцию **CreateWindow**, передав ей в качестве первого параметра указатель на строку “combobox”. Второй параметр должен быть указан как NULL. Например**:**
- static int ID_combo=5; static HWND hWndCombo; . . . hWndCombo=CreateWindow(**"combobox",**NULL, WS_CHILD|WS_VISIBLE|WS_CLIPSIBLINGS| **CBS_DROPDOWNLIST**, x,y,width,height,hWnd,(HMENU)**ID_combo**,hInst,NULL);
- При создании списка “combobox” указываются **специальные стили комбинированного списка**, символические имена которых имеют префикс **CBS_.**

Выпадающий список

- **Рассмотрим базовые стили комбинированного списка**
- Среди всех стилей комбинированного списка можно выделить три базовых:
- Стиль **CBS_SIMPLE** соответствует **списку с окном редактирования** (или, как его называют, окном выбора).
- Если список имеет стиль **CBS_DROPDOWN**, **в исходном состоянии он состоит из окна редактирования и расположенной справа пиктограммы со стрелкой** (кнопкой, предназначенной для отображения списка). Если нажать на эту пиктограмму левой клавишей мыши, **под окном редактирования появится список**.
- Стиль **CBS_DROPDOWNLIST** аналогичен стилю **CBS_DROPDOWN**, но **окно редактирования можно использовать только для просмотра выделенной строки**, а не для редактирования или ввода.
- **Рассмотрим сообщения от комбинированного списка, посылаемые родительскому окну**
- **Комбинированные списки посылают оконной процедуре родительского окна сообщения WM_COMMAND**. Значения параметров сообщения wParam и lParam такие же как и для кнопок. **Коды извещения для комбинированного списка** имеют символические имена с префиксом **CBN_**.

Выпадающий список

- **Для обработки сообщения от комбинированного списка оконная функция** родительского окна может содержать код следующего вида:
- case WM_COMMAND: { UINT idCtl=LOWORD(wParam); // идентификатор дочер. окна UINT code=HIWORD(wParam); // код уведомления HWND hChild=(HWND)lParam; // дескриптор дочер. окна If(idCtrl==ID_combo&&code==CBN_SELCHANGE) { // сделан выбор в комбинированном списке . . . } }; return 0; **Рассмотрим сообщения, посылаемые родительским окном комбинированным спискам**
- Для управления списком “combobox” **используется набор сообщений, аналогичный набору сообщений для списка “listbox” и редактора текста “edit”.** Функция **SendMessage**, посылающая сообщения списку, возвращает значение, которое зависит от выполняемой функции, или коды ошибок.
- В файле windows.h определены сообщения, специально предназначенные для работы со списком “combobox”. **Символические имена этих сообщений имеют префикс CB_.**

- **Для обработки сообщения от комбинированного списка оконная функция** родительского окна может содержать код следующего вида:
- case WM_COMMAND: { UINT idCtl=LOWORD(wParam); // идентификатор дочер. окна UINT code=HIWORD(wParam); // код уведомления HWND hChild=(HWND)lParam; // дескриптор дочер. окна If(idCtrl==ID_combo&&code==CBN_SELCHANGE) { // сделан выбор в комбинированном списке . . . } }; return 0; **Рассмотрим сообщения, посылаемые родительским окном комбинированным спискам**
- Для управления списком “combobox” **используется набор сообщений, аналогичный набору сообщений для списка “listbox” и редактора текста “edit”.** Функция **SendMessage**, посылающая сообщения списку, возвращает значение, которое зависит от выполняемой функции, или коды ошибок.
- В файле windows.h определены сообщения, специально предназначенные для работы со списком “combobox”. **Символические имена этих сообщений имеют префикс CB_.**

Полосы прокрутки

- **Для создания дочернего окна управления типа полоса прокрутки** приложение должно вызвать функцию **CreateWindow**, передав ей в качестве первого параметра указатель на строку “scrollbar”, например:
- static int ID_scroll=6; static HWND hWndScroll; . . . hWndScroll=CreateWindow(**"scrollbar"**,NULL, WS_CHILD|WS_VISIBLE|WS_CLIPSIBLINGS| **SBS_VERT|**, x,y,width,height,hWnd,(HMENU)**ID_scroll**,hInst,NULL); **В отличие от других дочерних окон управления**, посылающих родительскому окну сообщения, **полосы прокрутки не посылают сообщения WM_COMMAND**. Вместо этого **они, также как и полосы прокрутки окна, посылают ему сообщения WM_VSCROLL** и **WM_HSCROLL**.
- При обработке сообщений полос прокрутки **приложению с помощью параметра lParam может различать сообщения полос прокрутки окна и полос прокрутки – элементов управления.** Для полос прокрутки окна lParam равен NULL, а для полос – элементов управления он является дескриптором этих полос прокрутки, другими словами дескриптором дочернего окна
- Значения же **старшего и младшего слов параметра wParam для полос прокрутки окна и для полос прокрутки – дочерних окон имеют одинаковый смысл**.

Полосы прокрутки

- **Замечание.** Полосы прокрутки – элементы управления можно сделать любого размера. Для того чтобы **создать элементы управления с теми же стандартными размерами**, что и полосы прокрутки окна, можно использовать функцию **GetSystemMetrics**:
- вызов **GetSystemMetrics(SM_CYHSCROLL)** выдает высоту горизонтальной полосы прокрутки,
- а вызов **GetSystemMetrics(SM_CXVSCROLL)** – ширину вертикальной полосы прокрутки.
- Для полос прокрутки – элементов управления также **можно установить диапазон и текущее положение** с помощью тех же вызовов функция, что и для полос прокрутки окна:
- SetScrollRange(**hWndScroll**,**SB_CTL**,iNewMin,iNewMax,bRedraw);
  Полосы прокрутки – элементы управления могут обрабатывать сообщения клавиатуры, но только в том случае, если они имеют фокус ввода. Если необходимо, чтобы полоса прокрутки управления получала фокус ввода, когда на полосе прокрутки происходит щелчок мыши, то **следует включить идентификатор WS_TABSTOP** в параметр стиля дочернего окна управления при вызове функции **CreateWindow**.

Полосы прокрутки

- Приведем **пример обработки сообщений от полосы прокрутки – дочернего окна управления**. Пусть в окне hWnd создано дочернее окно управления hWndScroll – вертикальная линейка прокрутки, и для нее установлены диапазон и текущее положение бегунка:
- static int min_sb=1,max_sb=100,pos_sb=20; SetScrollRange(hWndScroll,SB_CTL,min_sb,max_sb,TRUE); SetScrollPos(hWndScroll,SB_CTL,pos_sb,TRUE); Рассмотрим фрагмент оконной функции, демонстрирующий возможную обработку действий пользователя с этой вертикальной линейкой прокрутки:
- case WM_VSCROLL: { // произведенное действие int nScrollCode=(int)LOWORD(wParam); // текущая позиция short int nPos=(short int)HIWORD(wParam); // дескриптор полосы просмотра или NULL HWND hwndScrollBar=(HWND)lParam; int old_pos_sb=pos_sb; 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 0l; } if(pos_sb<min_sb) pos_sb=min_sb; if(pos_sb>max_sb) pos_sb=max_sb; if(old_pos_sb!=pos_sb) SetScrollPos(hWndScroll,SB_CTL,pos_sb,TRUE); }; return 0l;

Дополнительные элементы

- Для упрощения создания Windows-приложений с интерфейсом пользователя, соответствующим **элегантному интерфейсу** оболочки системы Windows, фирма Microsoft разработала **библиотеку элементов управления общего пользования** (common control library).
- **Элементы управления главного окна** – элементы управления, обычно используемые в главном окне.

  Toolbar (панель инструментов) - Состоит из кнопок быстрого доступа.
  Tooltip (окно подсказки) - Обеспечивает пользователя быстрой подсказкой, отображая текст во всплывающем окне.
  Status bar (строка состояния) - Информационная строка, обычно размещаемая в нижней части окна приложения.

- **Составные диалоговые элементы управления** - элементы управления из списков свойств и мастеров

  Property page (станица свойств) - Немодальное диалоговое окно, используемое как одна страница в списке свойств или мастере (wizards).
  Property sheet (набор страниц свойств) - Набор из множества окон страниц свойств.

  Элементы управления Windows Explorer - элементы управления для построения приложений, похожих на Windows Explorer.

  Tree view (дерево просмотра) - Отображает иерархически элементизированный список (левая панель окна программы Windows Explorer).
  List view (список просмотра) - Отображает список элементов, идентифицируемых битовым образом и текстовыми данными (правая панель окна программы Windows Explorer).

Дополнительные элементы

- **Другие элементы управления**

  Animation (анимационное изображение) - Проигрывает анимационную последовательность для индикации длительной операции.
  Drag list (список, поддерживающий операции типа drag/drop) - Окна списка, поддерживающие простые операции drag/drop по отношению к себе и другим окнам типа Drag list. (Не drag/grop OLE-контейнер).
  Header (заголовок списка просмотра) - Отображает горизонтальные заголовки для столбцов (используется совместно со списком просмотра).
  Hot-Key (горячая клавиша) - Отображает результат операции определения клавиш активизации (горячих клавиш).
  Image list (список изображений) - Элемент управления для хранения набора растровых изображений (битовых образов, курсоров, значков), не являющийся окном.
  Progress bar (индикатор процесса) - Отображает динамику длительной операции как процент от выполненной задачи.
  Rich edit (усовершенствованный редактор) - Редактор, поддерживающий множество шрифтов и базовые возможности OLE-контейнера.
  Tab (набор закладок для выбора) - Отображает список закладок для выбора. Tabs используются в окне набора страниц свойств для выбора страницы свойств. Панель задач (task bar) Windows 95 – есть элемент управления Tab, использующий кнопки вместо закладок.
  Trackbar (окно с движком для выбора значения из диапазона) - Тип полосы прокрутки для выбора значения в заданном диапазоне.
  Up-Down (полоса прокрутки, связанная с окном редактирования для увеличения или уменьшения на 1 целочисленного значения) - Тип полосы прокрутки, состоящий из двух стрелок (но собственно без полосы для увеличения или уменьшения на 1 величины, находящейся в связанном поле редактирования).

Основы общих элементов управления

- **Каждый элемент управления общего пользования, за исключением списка изображений, реализован как класс окна**. С этой точки зрения, **элементы управления общего пользования похожи на предопределенные (стандартные) элементы управления** диалоговых окон, появившиеся в первой реализации системы Windows.
- Оба типа элементов управления строятся с помощью функции **CreateWindow**, настраиваются с использованием конкретных флагов стиля класса, управляются специфичными для данного класса сообщениями и приводятся к нужному состоянию с применением обычных API – вызовов, манипулирующих с окнами.
- Оба типа элементов управления также посылают уведомляющие сообщения родительскому окну, информируя обо всех происходящих событиях.
- **Разница между элементами управления общего пользования и предопределенными элементами управления состоит в том, какие сообщения они посылают для уведомления**. Предопределенные элементы управления посылают уведомляющие **сообщения WM_COMMAND**, в то время как элементы управления общего пользования (за некоторыми исключениями) посылают сообщения **WM_NOTIFY**.
- Хотя при поверхностном взгляде механизмы передачи отличаются, идея, лежащая в основе этих уведомляющих сообщений, одна и та же: **родительское окно может иметь возможность реагировать на все интересующие его события**.

Основы общих элементов управления

- Наиболее важно при работе с элементами управления обоих типов помнить, что **элемент управления – это окно, и все, что уже известно о манипулировании окнами, применимо при манипулировании элементами управлени**я. Также следует запомнить, что **все, что известно о работе с предопределенными диалоговыми элементами управления, относится и к работе с элементами управления общего пользования**.
- Фактически, только один элемент управления – усовершенствованный редактор – является расширенным окном редактирования, поддерживающим такой же базовый набор стилей управления, сообщений и уведомлений, как и оригинал – предопределенное окно редактирования (плюс некоторые новые возможности).
- Так же как и при работе с предопределенными элементами управления**, преимущества от использования элементов управления общего пользования** состоят в том, что, считая каждый из этих элементов “черным ящиком”, разработчик **получает множество возможностей с минимумом затрат с его стороны**.
- Ключевым моментом для работы с элементами управления общего пользования **является понимание управления и конфигурирование набора конкретных средств** таким образом, чтобы при минимальных затратах добиться нужного поведения и внешнего вида элемента управления. Узнав о возможностях, остается только добиться того, чтобы сделать элемент управления работающим на приложение.

Инициализация библиотек и элементов общего пользования

- **Различные описания, которые необходимы для использования элементов управления общего пользования (за исключением усовершенствованного редактора), определяются в файле commctrl.h.** Этот файл не является частью основной группы файлов, на которые имеет ссылки файл windows.h. Поэтому, любой исходный файл, ссылающийся на функции элементов общего пользования, типы данных и т.д., должен обязательно содержать следующую строку:
- #include <commctrl.h>
- Файл comctl32.lib представляет собой **библиотеку импорта** элементов управления общего пользования (ее следует подключить на этапе компоновки приложения), а файл comctl32.dll - это **динамически подключаемая библиотека**, в которой содержатся шаблоны и процедуры элементов управления (она будет использоваться во время работы приложения).
- **Перед использованием** какого-либо элемента управления общего пользования **программа должна вызвать функцию InitCommonControls** (как правило, **это делается в самом начале функции WinMain** приложения), которая регистрирует классы окон этих элементов, используя функцию RegisterClass.
- InitCommonControls();
- Функция **InitCommonControls** не имеет параметров и не возвращает никакого значения.

Инициализация библиотек и элементов общего пользования

- Ввиду размера и сложности **усовершенствованного редактора**, он располагается **в его собственной динамически подключаемой библиотеке** riched32.dll. Усовершенствованный редактор регистрирует сам себя при загрузке этой библиотеки, которую следует вызвать **посредством функции LoadLibrary** (как правило, **это делается в самом начале функции WinMain** приложения)
- LoadLibrary(“riched32.dll”); **Различные описания, которые необходимы для использования усовершенствованного редактора находятся в файле richedit.h, а связанные с OLE описания для него находятся в файле richole.h.**

Создание элементов общего пользования

- **Создание элементов управления общего пользования**
- **Наиболее общий путь создания окна элемента управления общего пользования состоит в вызове функции CreateWindow или CreateWindowEx** (функция **CreateWindowEx** идентична функции **CreateWindow**, с тем исключением, что она использует дополнительные стили, эти стили окна рассматриваются далее). Например, приведенный вызов строит панель инструментов:
- #define ID_TOOLBAR 1 . . . HWND hWndToolbar=CreateWindow(TOOLBARCLASSNAME,NULL, CCS_TOP|WS_CHILD|WS_VISIBLE|WS_BORDER|WS_CLIPSIBLINGS, 0,0,0,0,hWndParent,(HMENU)ID_TOOLBAR,hInstance,0);
- **Имя класса окна не задается в кавычках, поскольку это символическая константа**, определение которой зависит от набора символов, выбранного при построении программы. Для набора символов ANSI символическая константа **TOOLBARCLASSNAME** заменяется строкой “ToolbarWindow32”; для набора символов UNICODE символ “L” ставится перед этим именем (L“ToolbarWindow32”) для создания UNICODE-строки. **Все классы элементов управления общего пользования определяются этим способом.**
- **Чаще всего элементы управления общего пользования создаются как дочерние окна**, что определяется заданием флага **WS_CHILD** и установкой **описателя родительского окна** hWndParent.

Создание элементов общего пользования

- Как показано в примере, дочерние окна часто создаются с начальным местоположением (x,y) и размерами (cx,cy) равными нулю, а затем изменяют свой размер при изменении размеров родительского окна (т.е. когда родительское окно получает сообщение **WM_SIZE**).
- **Альтернативой вызову функции CreateWindow является вызов специализированной функции создания элемента управления**, которая обычно выполняет некоторую стандартную инициализацию.
- Примером специализированной функции создания элемента управления является функция **CreateToobarEx**, строящая панель инструментов, и добавляющие в нее кнопки.
- В других случаях, таких как набор страниц свойств, имя класса недоступно, поэтому требуется вызов специализированной функции: **PropertySheet** строит набор страниц свойств, а **CreatePropertySheetPage** строит индивидуальные страницы свойств.
- Список изображений строится вызовом функции **ImageList_Create** – специализированной функции, поскольку список изображений не является окном.

Создание элементов общего пользования

- Приведем **все имена классов элементов управления общего пользования и функции их создания**.
- **Элементы управления главного окна**

Tool bar	TOOLBARCLASSNAME	CreateToolbarEx
Tool tip	TOOLTIPS_CLASS	Нет
Status bar	STATUSCLASSNAME	CreateStatusWindow

Составные диалоговые элементы управления

Property page	Нет	CreatePropertySheetPage
Property sheet	Нет	PropertySheet

- **Элементы управления Windows Explorer**

Tree view	WC_TREEVIEW	Нет
List view	WC_LISTVIEW	Нет

- **Другие элементы управления**

Animation	ANIMATE_CLASS	Нет
Drag list	“listbox” или L“listbox”	MakeDragList
Header	WC_HEADER	Нет
Hot key	HOTKEY_CLASS	Нет
Image list	Нет	ImageList_Create
Progress bar	PROGRESS_CLASS	Нет
Rich edit	“RichEdit” илиL”RichEdit”	Нет
Tab	WC_TABCONTROL	Нет
Trackbar	TRACKBAR_CLASS	Нет
Up-Down	UPDOWN_CLASS	CreateUpDownControl

Посылка сообщений общим элементам управления

- **После создания окна общего элемента для управления его действиями ему посылаются сообщения**. Для этого требуется вызов функции **SendMessage** с ее традиционными четырьмя параметрами: дескриптор-окна получателя, идентификатор сообщения, значение wParam и lParam. Так же как существуют специфические флаги стилей общих элементов управления, так и существуют специфические сообщения.
- **Альтернативой вызовам функции SendMessage является использование наборов макросов языка C**, определенных в файле commctrl.h, которые получают специфичный для сообщения набор параметров, осуществляют необходимые преобразования, а затем вызывают функцию **SendMessage**. Возвращаемое значение также преобразуется к нужному типу, поскольку часто значение типа LRESULT, возвращаемое функцией **SendMessage**, не совпадает с ожидаемым типом возвращаемого значения.
- Несмотря на то, что макросы очень удобны и полезны, заголовочные файлы Windows, к сожалению, включают **определения лишь для половины элементов управления общего пользования**: для анимационного изображения, заголовка списка просмотра, списка просмотра, набора страниц свойств, дерева просмотра и набора закладок для выбора.

Уведомляющие сообщения от общих элементов управления

- Как и предопределенные элементы управления, **общие элементы управления посылают своему родительскому окну уведомляющие сообщения**. Уведомляющие сообщения информируют родительское окно о том, что что-то произошло с окном элемента управления.
- В отличие от предопределенных элементов управления, которые посылают уведомления как сообщения **WM_COMMAND**, **общие элементы управления обычно посылают уведомления как сообщения WM_NOTIFY.** Таким образом, если добавить общий элемент управления к существующему уже коду приложения, то **смешивания обработки уведомляющих сообщений** от предопределенных элементов управления и общих элементов в программе не произойдет. Сообщения **WM_NOTIFY** также предотвращают **путаницу с уведомляющими сообщениями от меню**, которые тоже выражаются в виде сообщений **WM_COMMAND**.
- **Вместе с сообщением WM_NOTIFY через параметры сообщения приходит необходимая приложению информация**. Через параметр wParam передается **идентификатор элемента управления**, пославшего это сообщение. Параметр lParam содержит указатель на структуру типа **NMHDR**, в которой содержатся **код уведомления и другая дополнительная информация**.

Уведомляющие сообщения от общих элементов управления

- Приведем **фрагмент программы, в котором демонстрируется последовательность действий при обработке события WM_NOTIFY**:
- case WM_NOTIFY:

```
int idCommonCtrl=(int)wParam;              // идентификатор элемента управления

LPNMHDR pNMH=(LPNMHDR)lParam;        // указатель на структуру              int iCode= pNMH->code;              // код уведомления
HWND hWndFrom= pNMH->hwndFrom;       // дескриптор элемента управления

if(idCommonCtrl== ID_TOOLBAR)
{           // обработка сообщения от элемента управления                    // с идентификатором ID_TOOLBAR                . . .          }          return 0l;
```

Однако не все уведомления общих элементов приходят как сообщения WM_NOTIFY.

- В частности, **панель инструментов**, использующая сообщения **WM_NOTIFY** для большинства уведомлений, посылает сообщения **WM_COMMAND**, **когда нажимается кнопка**. Поскольку панель инструментов обычно используется для поддержки выбора из меню, тот же код обработки сообщений **WM_COMMAND**, что для поддержки меню и быстрых клавиш, будет обрабатывать сообщения от панели инструментов.
- И еще одно исключение – «**полоса прокрутки» Track bar**, связанная с окном редактирования для изменения значения, которая также посылает сообщения **WM_VSCROOL** и **WM_HSCROLL** при нажатии на стрелки.

Уведомляющие сообщения от общих элементов управления

- Хотя каждый общий элемент управления имеет свой собственный набор кодов уведомления, существует **общий набор уведомлений**:

  NM_CLICK – пользовательсделал щелчок левой кнопкой мыши.
- **NM_DBLCLK** - пользовательсделал двойной щелчок левой кнопкой мыши.
- **NM_KILLFOCUS** – элемент управления потерял фокус ввода.
- **NM_OUTOFMEMORY** – ошибка нехватки памяти.
- **NM_RCLICK** - пользовательсделал щелчок правой кнопкой мыши.
- **NM_RDBLCLK** - пользовательсделал двойной щелчок правой кнопкой мыши.
- **NM_RETURN** – пользователь нажал клавишу <Enter>.
- **NM_SETFOCUS** – элемент управления получил фокус ввода.
- **Не все общие элементы управления обязательно посылают каждое из этих уведомляющих сообщений.** Например, набор закладок для выбора не посылает уведомлений, связанных с изменением фокуса ввода.
- □ При работе с каждым элементом управления общего пользования следует разобраться с тем, какие уведомляющие события он посылает.