Модуль №3

Занятие №5

Версия 1.0.1

План занятия:

1. Повторение пройденного материала.
2. Общий элемент управления «регулятор» (Slider Control).
3. Сообщения регулятора.
4. Общий элемент управления «счётчик» (Spin Control).
5. Сообщения счётчика.
6. Практическая часть.
7. Подведение итогов.
8. Домашнее задание.

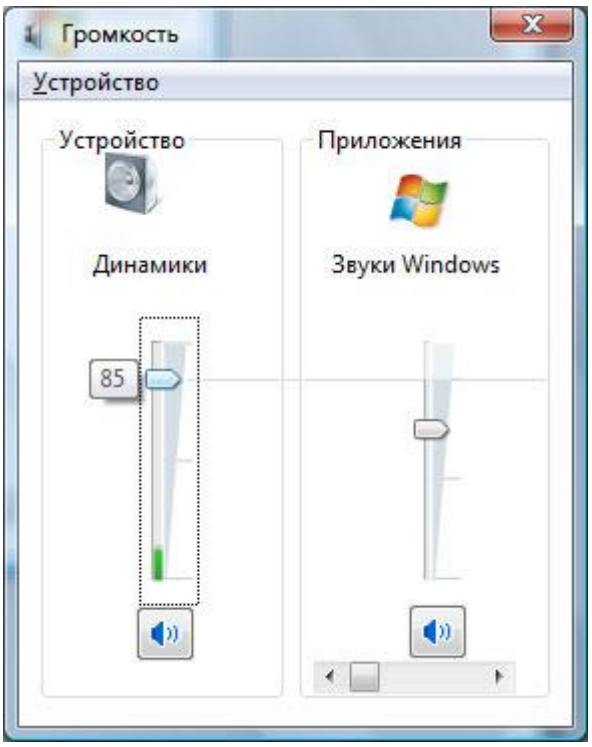
1. Повторение пройденного материала

Данное занятие необходимо начать с краткого повторения материала предыдущего занятия. При общении со слушателями можно использовать следующие контрольные вопросы:

1) Что такое распаковщики сообщений? Какими преимуществами они обладают?
2) В чем состоит главное отличие общих элементов управления от базовых элементов управления?
3) Для чего применяется элемент управления **Progress Control**?
4) Какое сообщение необходимо отправить индикатору, чтобы установить ему интервал (нижнюю и верхнюю границу)?

5) Какие макросы позволяют упаковать дополнительную информацию в параметры **WPARAM** и **LPARAM**?
6) С помощью какого сообщения можно установить шаг приращения для индикатора?
7) Какими способами можно изменить текущее состояние индикатора?
8) Как установить цвет фона индикатора?
9) Как установить цвет самого индикатора (цвет заполняемых прямоугольников)?
10) Как программно создать **Progress Control**?

2. Общий элемент управления «регулятор» (Slider Control)



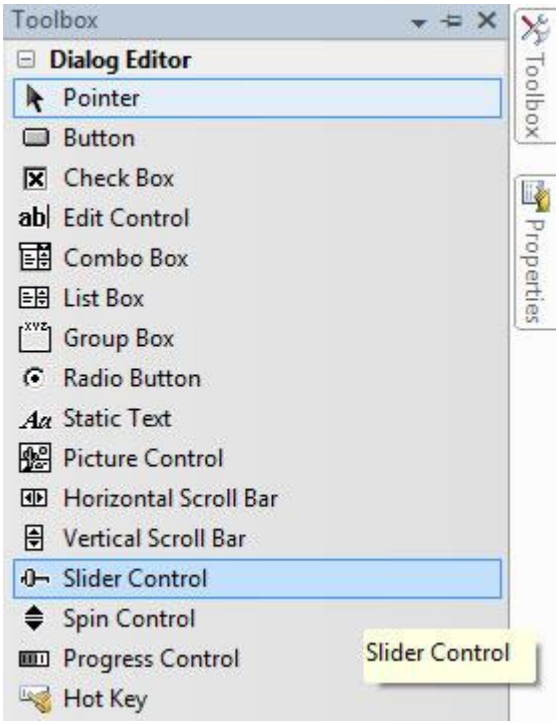
Элемент управления **Slider Control** (ползунок или регулятор), который ранее назывался **Trackbar**, представляет собой окно с линейкой и

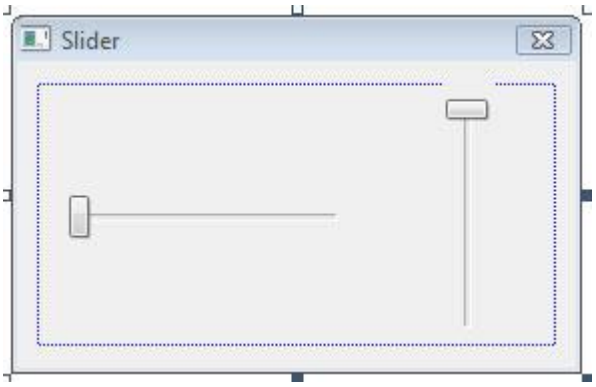
перемещаемым по ней ползунком. Примером этого элемента управления является «Регулятор громкости» операционной системы Windows. Подобный регулятор дает возможность пользователю выбирать дискретные значения в заданном диапазоне.

Создать **Slider Control** на форме диалога можно, как обычно, двумя способами:

- с помощью средств интегрированной среды разработки **Microsoft Visual Studio**;
- посредством вызова функции **CreateWindowEx**.

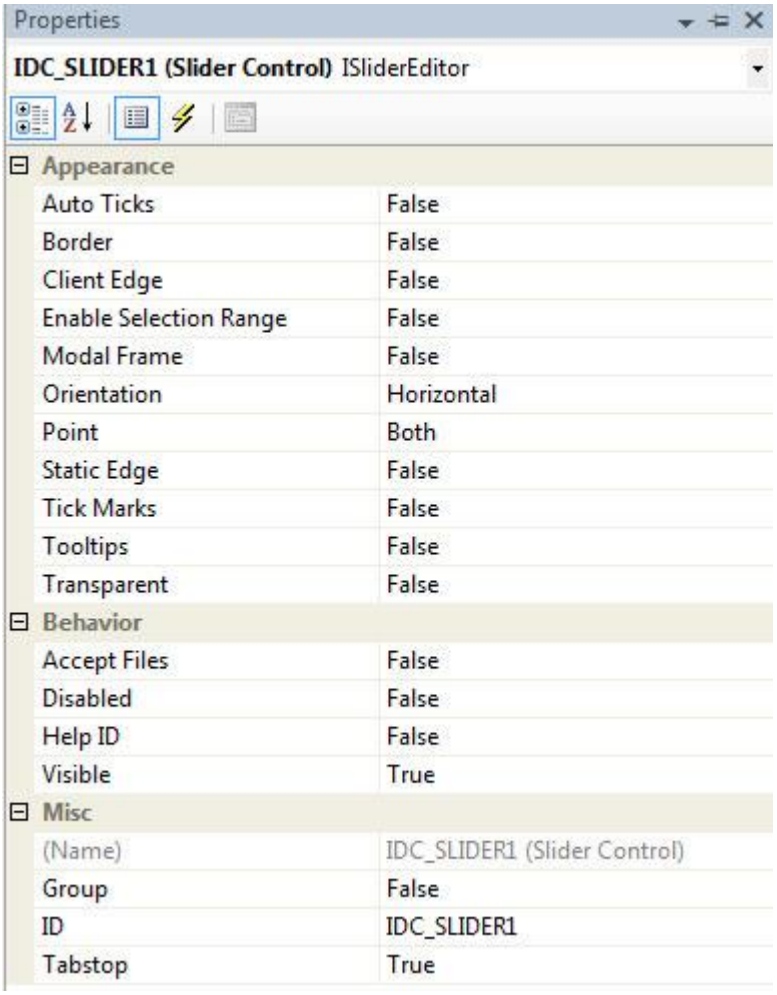
При первом способе необходимо определить **Slider Control** в шаблоне диалогового окна на языке описания шаблона диалога. Для этого следует активизировать окно **Toolbox** и «перетащить» регулятор на форму диалога.





После размещения регулятора на форме диалога ему назначается идентификатор (например, **IDC_SLIDER1**), который впоследствии можно изменить на идентификатор, отражающий семантику ресурса.

Необходимо ознакомить слушателей с некоторыми свойствами элемента управления **Slider Control**:



- свойство **Orientation** позволяет выбрать ориентацию элемента управления: **Horizontal** (по умолчанию) или **Vertical**;
- свойство **Point** предназначено для выбора стиля размещения меток: **Both** (по умолчанию метки размещаются с обеих сторон), **Top/Left** или **Bottom/Right**;
- свойство **Tick Marks** задаёт наличие или отсутствие меток;
- свойство **Auto Ticks** определяет, будут ли метки для всех значений в заданном диапазоне значений или только для начального и конечного положений ползунка;
- свойство **Border** определяет наличие рамки у элемента управления;
- свойство **Tooltips** включает поддержку всплывающих подсказок, отображающих текущую позицию ползунка.

3. Сообщения регулятора

Ознакомить слушателей с наиболее часто используемыми сообщениями для управления регулятором.

Код сообщения	wParam	lParam	Описание
TBM_GETPOS	0	0	Возвращает текущую позицию ползунка
TBM_SETPOS	fRedraw	newPos	Устанавливает новую позицию ползунка. Если **fRedraw** равно TRUE, регулятор перерисовывается после этого.
TBM_SETRANGE	fRedraw	MAKELPARAM (wMin, wMax)	Устанавливает диапазон регулятора
TBM_SETTICFREQ	wFreq	0	Устанавливает шаг меток
TBM_SETLINESIZE	0	nLineSize	Устанавливает размер «строки»
TBM_SETPAGESIZE	0	nPageSize	Устанавливает размер «страницы»

Необходимо дать подробный комментарий по каждому из сообщений, приведенных в таблице. В частности, отметить, что с элементом управления **Slider Control** связан внутренний счетчик, определяющий его поведение. Счетчик имеет минимальное значение (по умолчанию 0) и максимальное значение (по умолчанию 100). При этом существует возможность изменить диапазон ползунка, послав ему сообщение **TBM_SETRANGE**.

Текущее состояние счетчика однозначно связано с текущей позицией ползунка. Перемещать ползунок по линейке регулятора можно как с помощью мыши, так и с помощью клавиатуры. Второй вариант работы предполагает, что регулятор имеет фокус ввода.

Минимальный интервал («строка»), на который можно изменить состояние регулятора с помощью клавиш управления курсором, по умолчанию равен единице. Этот интервал можно задать, послав регулятору сообщение **TBM_SETLINESIZE**.

Более крупный интервал («страница»), на который можно изменить состояние регулятора с помощью клавиш **Page Up** или **Page Down**, ра-

вен одной пятой части диапазона регулятора. Этот интервал можно задать, послав регулятору сообщение **TBM_SETPAGESIZE**.

Следует отметить, если регулятор создан со свойством **Auto Ticks**, то линейка имеет метки во всем диапазоне значений с шагом **wFreq**, который по умолчанию равен единице. Существует возможность изменить этот шаг, послав регулятору сообщение **TBM_SETTICFREQ**.

Акцентировать внимание слушателей на том, что ползунок уведомляет свое родительское окно (диалог) о действиях пользователя, посылая сообщение **WM_HSCROLL** или **WM_VSCROLL** — в зависимости от ориентации элемента управления (**Horizontal** или **Vertical**). В любом случае в младшем слове параметра **wParam** содержится код уведомления, а параметр **lParam** содержит дескриптор ползунка.

Возможные коды уведомления приведены в следующей таблице.

Код уведомления	**Интерпретация**
TB_LINEUP	Нажата клавиша «стрелка влево» (**VK_LEFT**) или клавиша «стрелка вверх» (**VK_UP**)
TB_LINEDOWN	Нажата клавиша «стрелка вправо» (**VK_RIGHT**) или клавиша «стрелка вниз» (**VK_DOWN**)
TB_PAGEUP	Нажата клавиша «Page Up» (**VK_PRIOR**) или щелчок мышью на линейке левее или выше ползунка
TB_ PAGEDOWN	Нажата клавиша «Page Down» (**VK_NEXT**) или щелчок мышью на линейке правее или ниже ползунка
TB_THUMBPOSITION	Отпущена кнопка мыши после перемещения ползунка (это сообщение следует всегда после сообщения **TB_THUMBTRACK**)
TB_THUMBTRACK	Ползунок перемещается с помощью мыши
TB_TOP	Нажата клавиша «Home» (**VK_HOME**) — ползунок устанавливается в крайнее левое (верхнее) положение, соответствующее значению **wMin**
TB_BOTTOM	Нажата клавиша «End» (**VK_END**) — ползунок устанавливается в крайнее правое (нижнее) положение, соответствующее значению **wMax**

Следует подчеркнуть, что для кодов **TB_THUMBPOSITION** и **TB_THUMBTRACK** старшее слово параметра **wParam** содержит позицию ползунка.

Акцентировать внимание слушателей на том, что очень часто приложение может обойтись без обработки этих сообщений, так как с помощью управляющего сообщения **TBM_GETPOS** можно легко получить текущую позицию ползунка, и в большинстве случаев этого оказывается достаточно.

В качестве примера использования регулятора продемонстрировать слушателям следующее приложение, в котором с помощью ползунков изменяются размеры главного окна.

```
                                    // header.h

#pragma once

#include <windows.h>
#include <windowsX.h>
#include <tchar.h>
#include <commctrl.h>
#include "resource.h"

#pragma comment(lib,"comctl32")

#define MIN 0
#define MAX 400
                                // SliderControlDlg.h

#pragma once
#include "header.h"

class CSliderControlDlg
{
public:
      CSliderControlDlg(void);
public:
      static BOOL CALLBACK DlgProc(HWND hWnd, UINT mes, WPARAM wp, LPARAM lp);
      static CSliderControlDlg* ptr;
      BOOL Cls_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);
      void Cls_OnHScroll(HWND hwnd, HWND hwndCtl, UINT code, int pos);
      void Cls_OnVScroll(HWND hwnd, HWND hwndCtl, UINT code, int pos);
      void Cls_OnClose(HWND hwnd);
      HWND hDialog, hHorizontalSlider, hVerticalSlider;
};


                               // SliderControlDlg.cpp

#include "SliderControlDlg.h"

CSliderControlDlg* CSliderControlDlg::ptr = NULL;
```

```
CSliderControlDlg::CSliderControlDlg(void)
{
	ptr = this;
}

void CSliderControlDlg::Cls_OnClose(HWND hwnd)
{
	EndDialog(hwnd, 0);
}

BOOL CSliderControlDlg::Cls_OnInitDialog(HWND hwnd, HWND hwndFocus,
                                         LPARAM lParam)
{
	hDialog = hwnd;
	hHorizontalSlider = GetDlgItem(hDialog, IDC_SLIDER1);
	hVerticalSlider = GetDlgItem(hDialog, IDC_SLIDER2);
	SendMessage(hHorizontalSlider, TBM_SETRANGE, TRUE,
		    MAKELPARAM(MIN, MAX));
	SendMessage(hVerticalSlider, TBM_SETRANGE, TRUE, MAKELPARAM(MIN, MAX));
	return TRUE;
}

void CSliderControlDlg::Cls_OnHScroll(HWND hwnd, HWND hwndCtl, UINT code,
                                      int pos)
{
	static int nOldPosition = 0;
	int nCurrrentPosition = SendMessage(hwndCtl, TBM_GETPOS, TRUE,
                                MAKELPARAM(MIN, MAX));
	RECT rect;
	GetWindowRect(hwnd, &rect);
	switch(code)
	{
		case TB_BOTTOM:
			// Нажата клавиша «End» (VK_END) – ползунок  устанавливается
			// в крайнее правое положение
			rect.right += MAX - nOldPosition;
			break;
		case TB_TOP:
			// Нажата клавиша «Home» (VK_HOME) – ползунок
			// устанавливается в крайнее левое положение
			rect.right -= nOldPosition - MIN;
			break;
		case TB_LINEUP:
			// Нажата клавиша «стрелка влево» (VK_LEFT)
			if(nCurrrentPosition > MIN) rect.right--;
			break;
		case TB_LINEDOWN:
			// Нажата клавиша «стрелка вправо» (VK_RIGHT)
			if(nCurrrentPosition < MAX) rect.right++;
			break;
		case TB_PAGEDOWN:
		case TB_PAGEUP:
		case TB_THUMBPOSITION:
			// Отпущена кнопка мыши после перемещения ползунка
		case TB_THUMBTRACK:
			// Ползунок перемещается с помощью мыши
			rect.right += nCurrrentPosition - nOldPosition;
			break;
	}
```

```
      nOldPosition = nCurrrentPosition;
      MoveWindow(hwnd, rect.left, rect.top, rect.right - rect.left,
                  rect.bottom - rect.top, 1);
}

void CSliderControlDlg::Cls_OnVScroll(HWND hwnd, HWND hwndCtl, UINT code,
                                      int pos)
{
      static int nOldPosition = 0;
      int nCurrrentPosition = SendMessage(hwndCtl, TBM_GETPOS, TRUE,
                              MAKELPARAM(MIN, MAX));
      RECT rect;
      GetWindowRect(hwnd, &rect);
      switch(code)
      {
      case TB_BOTTOM:
            // Нажата клавиша «End» (VK_END) – ползунок  устанавливается в
            // крайнее нижнее положение
            rect.bottom += MAX - nOldPosition;
            break;
      case TB_TOP:
            // Нажата клавиша «Home» (VK HOME) – ползунок устанавливается в
            // крайнее верхнее положение
            rect.bottom -= nOldPosition - MIN;
            break;
      case TB_LINEUP:
            // Нажата клавиша «стрелка вверх» (VK_UP)
            if(nCurrrentPosition > MIN) rect.bottom--;
            break;
      case TB_LINEDOWN:
            // Нажата клавиша «стрелка вниз» (VK_DOWN)
            if(nCurrrentPosition < MAX) rect.bottom++;
            break;
      case TB_PAGEDOWN:
      case TB_PAGEUP:
      case TB_THUMBPOSITION:
            // Отпущена кнопка мыши после перемещения ползунка
      case TB_THUMBTRACK:
            // Ползунок перемещается с помощью мыши
            rect.bottom += nCurrrentPosition - nOldPosition;
            break;
      }
      nOldPosition = nCurrrentPosition;
      MoveWindow(hwnd, rect.left, rect.top, rect.right - rect.left,
                  rect.bottom - rect.top, 1);
}

BOOL CALLBACK CSliderControlDlg::DlgProc(HWND hwnd, UINT message,
                                         WPARAM wParam, LPARAM lParam)
{
      switch(message)
      {
            HANDLE_MSG(hwnd, WM_CLOSE, ptr->Cls_OnClose);
            HANDLE_MSG(hwnd, WM_INITDIALOG, ptr->Cls_OnInitDialog);
            HANDLE_MSG(hwnd, WM_HSCROLL, ptr->Cls_OnHScroll);
            HANDLE_MSG(hwnd, WM_VSCROLL, ptr->Cls_OnVScroll);
      }
      return FALSE;
}
```

```
                              // SliderControlDlg.cpp

#include "SliderControlDlg.h"

int WINAPI _tWinMain(HINSTANCE hInst, HINSTANCE hPrev, LPTSTR lpszCmdLine,
                     int nCmdShow)
{
      INITCOMMONCONTROLSEX icc = {sizeof(INITCOMMONCONTROLSEX)};
      icc.dwICC = ICC_WIN95_CLASSES;
      InitCommonControlsEx(&icc);
      CSliderControlDlg dlg;
      return DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                       CSliderControlDlg::DlgProc);
}
```

Альтернативный способ создания ползунка - использование функции **CreateWindowEx**. В этом случае во втором параметре функции передается имя предопределенного оконного класса – **TRACKBAR_CLASS**.

В качестве примера, демонстрирующего программный способ создания ползунка, привести следующий фрагмент кода:

```
HWND hSlider = CreateWindowEx(0, TRACKBAR_CLASS, NULL, WS_CHILD | WS_VISIBLE,
                              LEFT, TOP, WIDTH, HEIGHT, hwndParent, NULL,
                              GetModuleHandle(NULL), NULL);
```

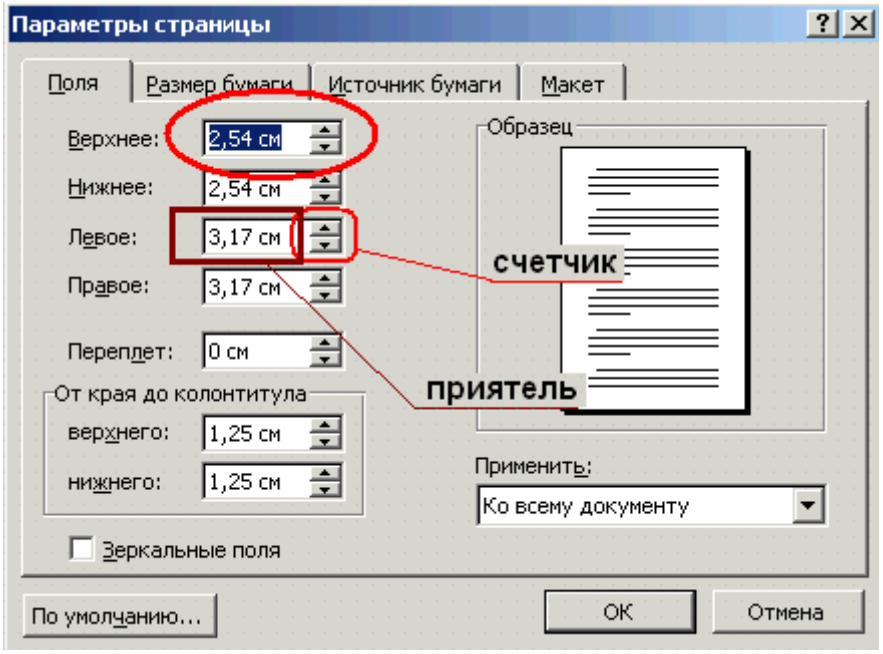
Кроме стилей **WS_CHILD** и **WS_VISIBLE** можно также задавать дополнительные стили, определяющие внешний вид элемента управления.

Стиль	Описание
TBS_HORZ	Линейка имеет горизонтальную ориентацию (стиль по умолчанию)
TBS_VERT	Линейка имеет вертикальную ориентацию
TBS_AUTOTICKS	Линейка имеет метки для всех значений в заданном диапазоне значений. Без этого стиля линейка может иметь метки только для начального и конечного положений ползунка
TBS_NOTICKS	Стиль исключает отображение каких-либо меток, в том числе и для начального и конечного положений ползунка
TBS_RIGHT	Элемент отображает метки справа (снизу) от линейки (стиль используется по умолчанию)

Стиль	Описание
TBS_LEFT	Элемент отображает метки слева (сверху) от линейки
TBS_BOTH	Элемент отображает метки с обеих сторон
TBS_TOOLTIPS	Поддерживается всплывающая подсказка, отображающая текущую позицию ползунка

4. Общий элемент управления «счётчик» (Spin Control)

Элемент управления **Spin Control** (счётчик) реализован как две кнопки со стрелками, с помощью которых можно увеличивать, или уменьшать некоторое числовое значение. Значение, связанное со счётчиком, называется его **текущей позицией**.



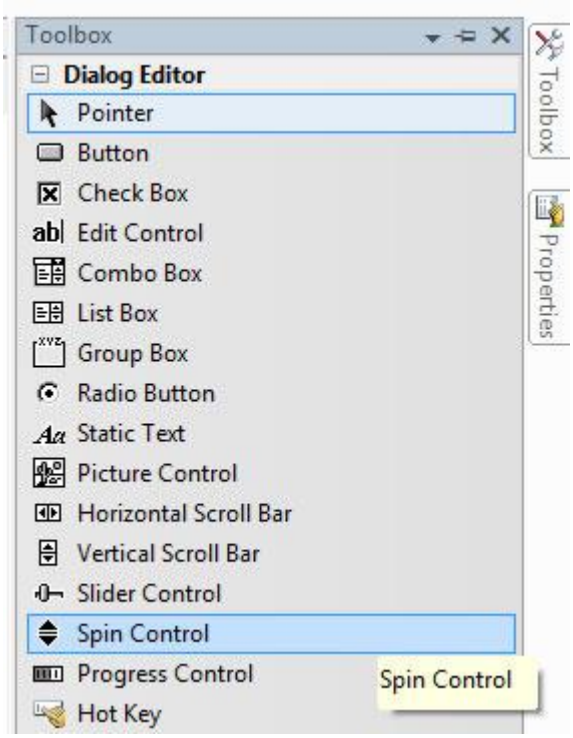
Кроме того, счётчик можно ассоциировать с другим элементом управления, называемым **приятельским окном** (**buddy window**). Чаще всего таким окном является текстовое поле. Комбинацию счётчика с текстовым полем называют также **полем с прокруткой**. Поле с прокруткой

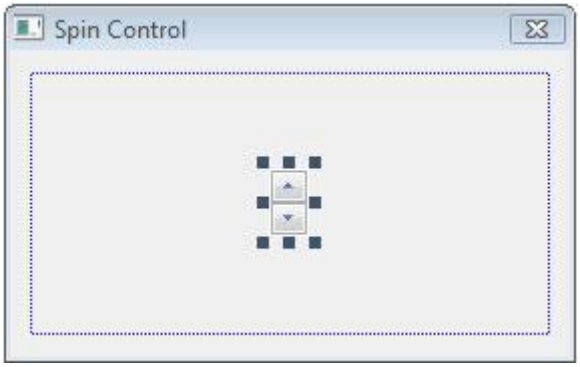
воспринимается пользователем как единый элемент управления. Содержимое текстового поля в таком элементе отображает текущую позицию счётчика.

Счётчик можно создать несколькими способами:

- при помощи редактора диалоговых окон на этапе визуального проектирования формы диалога;
- посредством вызова функции **CreateWindowEx**;
- посредством вызова функции **CreateUpDownControl**.

При первом способе необходимо определить **Spin Control** в шаблоне диалогового окна на языке описания шаблона диалога. Для этого следует активизировать окно **Toolbox** и «перетащить» счётчик на форму диалога.





После размещения спина на форме диалога ему назначается идентификатор (например, **IDC_SPIN1**), который впоследствии можно изменить на идентификатор, отражающий семантику ресурса.

Необходимо ознакомить слушателей с некоторыми свойствами элемента управления **Spin Control**.

- Свойство **Orientation** предназначено для выбора ориентации элемента управления: **Vertical** (по умолчанию) или **Horizontal**.
- Выпадающий список **Alignment** позволяет выбрать один из трех стилей размещения счётчика:
  - **Unattached** – счётчик располагается рядом с «приятелем»;
  - **Left** - счётчик располагается в левой части «приятеля», уменьшая тем самым его клиентскую область;
  - **Right** - счётчик располагается в правой части «приятеля», уменьшая тем самым его клиентскую область.
- Свойство **Auto Buddy** обеспечивает автоматический выбор в качестве «приятельского окна» ближайшего предыдущего элемента управления в файле описания ресурсов.

- Свойство **Set Buddy Integer** вместе с предыдущим свойством определяет синхронную работу счётчика и «приятеля»: любое изменение позиции счётчика сразу отображается в ассоциированном окне. Аналогично при вводе в «приятельское окно» допустимого целого числа устанавливается новая позиция счётчика.
- Свойство **No Thousands** позволяет вставлять пробел после каждых трех цифр в изображении десятичного числа.
- Свойство **Wrap** определяет работу счётчика – при истинном значении этого свойства счётчик работает как циклический, т.е. после максимального значения текущим становится минимальное, и наоборот.
- Свойство **Arrow Keys** поддерживает управление счётчиком с помощью клавиш управления курсором.

Таким образом, при создании поля с прокруткой необходимо поместить счётчик на форму диалога сразу вслед за размещением «приятеля» (текстового поля). Кроме того, свойствам **Auto Buddy** и **Set Buddy Integer** следует установить значение **True**.

5. Сообщения счётчика

Ознакомить слушателей с наиболее часто используемыми сообщениями для управления счётчиком.

Код сообщения	wParam	lParam	Описание
UDM_GETPOS32	0	(LPBOOL) pfError	Возвращает текущую позицию счетчика. **pfError** — указатель на булеву переменную, которая получает значение FALSE, если значение успешно получено, и TRUE — в случае ошибки.
UDM_SETPOS32	0	nPos	Устанавливает новую позицию счетчика.

Код сообщения	wParam	lParam	Описание
UDM_SETBASE	nBase	0	Устанавливает систему счисления (десятичную или шестнадцатеричную).
UDM_SETRANGE32	iLow	iHigh	Устанавливает минимальную и максимальную позиции для счетчика.
UDM_SETACCEL	nAccels	(LPUDACCEL) pAccels	Устанавливает правило, по которому будет осуществляться приращение счетчика. Параметр **nAccels** - задает количество структур, участвующих в определении правила, из массива **pAccels**.
UDM_GETACCEL	nAccels	(LPUDACCEL) pAccels	Позволяет узнать информацию о приращениях счетчика. Параметр **nAccels** - запрашиваемое количество структур, **pAccels** - указатель на массив структур для получения информации о приращениях счетчика. Возвращаемое значение - реальное количество полученных структур типа **UDACCEL**.
UDM_SETBUDDY	(HWND) hwndBuddy	0	Устанавливает «приятеля» для счетчика.

Следует отметить, если не определить диапазон счётчика при помощи сообщения **UDM_SETRANGE32**, то будет использоваться диапазон по умолчанию со значениями **nLower**=100 и **nUpper**=0.

Акцентировать внимание слушателей на том, что при нажатии одной из стрелок **Spin Control** посылает своему родительскому окну сообщение **WM_VSCROLL** или **WM_HSCROLL** (в зависимости от ориентации счётчика), в котором младшее слово параметра **wParam** содержит код **SB_THUMBPOSITION**. Кроме того, счетчик посылает уведомляющее сообщение **UDN_DELTAPOS** в форме сообщения **WM_NOTIFY**.

Обычно в приложении нет необходимости обрабатывать все сообщения. Часто бывает достаточно получить текущую позицию счетчика, обрабатывая сообщение **WM_VSCROLL** или **WM_HSCROLL**. Это можно сделать, отправив счётчику сообщение **UDM_GETPOS32**.

При непосредственном клавиатурном вводе нового числа в «приятельское окно» элемент **Edit Control** посылает диалогу сообщение **WM_COMMAND** с кодом уведомления **EN_CHANGE**. Для немедленной реакции приложения на изменившуюся текущую позицию счётчика следует предусмотреть обработку этого сообщения.

В качестве примера использования счётчика продемонстрировать слушателям следующее приложение.

```
                                    // header.h

#pragma once

#include <windows.h>
#include <windowsX.h>
#include <tchar.h>
#include <commctrl.h>
#include "resource.h"

#pragma comment(lib,"comctl32")

                                // SpinControlDlg.h


#pragma once
#include "header.h"

class CSpinControlDlg
{
public:
      CSpinControlDlg(void);
public:
      static BOOL CALLBACK DlgProc(HWND hWnd, UINT mes, WPARAM wp, LPARAM lp);
      static CSpinControlDlg* ptr;
      BOOL Cls_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam);
      void Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify);
      void Cls_OnTimer(HWND hwnd, UINT id);
      void Cls_OnClose(HWND hwnd);
      HWND hDialog, hSpin, hEdit1, hEdit2;
      TCHAR text[300];
      int count;
};

                               // SpinControlDlg.cpp

#include "SpinControlDlg.h"

CSpinControlDlg* CSpinControlDlg::ptr = NULL;

CSpinControlDlg::CSpinControlDlg(void)
{
      ptr = this;
}
```

```
void CSpinControlDlg::Cls_OnClose(HWND hwnd)
{
	EndDialog(hwnd, 0);
}


BOOL CSpinControlDlg::Cls_OnInitDialog(HWND hwnd, HWND hwndFocus,
                                       LPARAM lParam)
{
	hDialog = hwnd;
	hSpin = GetDlgItem(hDialog, IDC_SPIN1);
	hEdit1 = GetDlgItem(hDialog, IDC_EDIT1);
	hEdit2 = GetDlgItem(hDialog, IDC_EDIT2);

	// установим диапазон счетчика
	SendMessage(hSpin, UDM_SETRANGE32, 100, 10000);

	// зададим правило приращения
	UDACCEL pAcceleration[3] = {{1,10}, {3,100}, {5,500}};
	SendMessage(hSpin, UDM_SETACCEL, 3, LPARAM(pAcceleration));

	// установим приятеля
	SendMessage(hSpin, UDM_SETBUDDY, WPARAM(hEdit2), 0);

	SetWindowText(hEdit2, TEXT("100"));
	SetWindowText(hEdit1, TEXT("Общий элемент управления Spin Control"));
	return TRUE;
}

void CSpinControlDlg::Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl,
                                    UINT codeNotify)
{
	if(id == IDC_BUTTON2)
	{
		// Получим текущую позицию счетчика
		int nTime = SendMessage(hSpin, UDM_GETPOS32, 0, 0);
		SetTimer(hwnd, 1, nTime, NULL);
		// Получим текст с Edit Control
		GetWindowText(hEdit1, text, 300);
		count = 1;
	}
}

void CSpinControlDlg::Cls_OnTimer(HWND hwnd, UINT id)
{
	TCHAR copytext[300] = {0};
	if (count <= _tcslen(text)) //вывели весь текст?
	{
		//нет, тогда добавляем к заголовку один символ
		_tcsncpy(copytext, text, count++);
		//выводим текст в заголовок главного окна
		SetWindowText(hwnd, copytext);
	}
	else
	{
		//да, удаляем таймер
		KillTimer(hwnd, 1);
	}
}
```

```
BOOL CALLBACK CSpinControlDlg::DlgProc(HWND hwnd, UINT message,
                                       WPARAM wParam, LPARAM lParam)
{
      switch(message)
      {
            HANDLE_MSG(hwnd, WM_CLOSE, ptr->Cls_OnClose);
            HANDLE_MSG(hwnd, WM_INITDIALOG, ptr->Cls_OnInitDialog);
            HANDLE_MSG(hwnd, WM_COMMAND, ptr->Cls_OnCommand);
            HANDLE_MSG(hwnd, WM_TIMER, ptr->Cls_OnTimer);
      }
      return FALSE;

}
                                  // SpinControl.cpp

#include "SpinControlDlg.h"

int WINAPI _tWinMain(HINSTANCE hInst, HINSTANCE hPrev, LPTSTR lpszCmdLine,
                     int nCmdShow)
{

      INITCOMMONCONTROLSEX icc = {sizeof(INITCOMMONCONTROLSEX)};
      icc.dwICC = ICC_WIN95_CLASSES;
      InitCommonControlsEx(&icc);
      CSpinControlDlg dlg;
      return DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1), NULL,
                       CSpinControlDlg::DlgProc);

}
```

Альтернативный способ создания счётчика - использование функции **CreateWindowEx**. В этом случае во втором параметре функции передается имя предопределенного оконного класса – **UPDOWN_CLASS**.

В качестве примера, демонстрирующего программный способ создания счётчика, привести следующий фрагмент кода:

```
HWND hSpin = CreateWindowEx(0, UPDOWN_CLASS, NULL, WS_CHILD | WS_VISIBLE,
                            LEFT, TOP, WIDTH, HEIGHT, hwndParent, NULL,
                            GetModuleHandle(NULL), NULL);
```

Кроме стилей **WS_CHILD** и **WS_VISIBLE** можно также задавать дополнительные стили, определяющие внешний вид элемента управления.

Стиль	Описание
UDS_HORZ	Указывает на горизонтальную ориентацию счетчика (при отсутствии данного стиля ориентация вертикальная).
UDS_WRAP	Задает циклический переход от верхней границы к нижней (и наоборот) при достижении максимального значения верхней (нижней) границы.
UDS_ARROWKEYS	Разрешает использование клавиш управления курсором для изменения значения счетчика.
UDS_SETBUDDYINT	Указывает, что при изменении значения счетчика необходимо изменить значение «приятеля».
UDS_NOTHOUSANDS	Снимает разделитель между тысячами, к примеру, 1000000 или 1 000 000.
UDS_AUTOBUDDY	Позволяет автоматически выбирать «приятеля» из существующего списка окон.
UDS_ALIGNRIGHT	Счётчик будет находиться справа от «приятеля».
UDS_ALIGNLEFT	Счетчик будет находиться слева от «приятеля».

Существует ещё один программный способ создания счётчика – вызов функции API **CreateUpDownControl**, которая создаёт счётчик, определяет его минимальную, максимальную и текущую позиции, а также приятельское окно.

```
HWND CreateUpDownControl(
    DWORD dwStyle, // стили элемента управления
    int x, // клиентская координата X левого верхнего угла
    int y, // клиентская координата Y левого верхнего угла
    int cx, // ширина элемента управления
    int cy, // высота элемента управления
    HWND hParent, // дескриптор родительского окна
    int nID, // идентификатор элемента управления
    HINSTANCE hInst, // дескриптор приложения
    HWND hBuddy, // дескриптор «приятеля»
    int nUpper, // верхняя граница
    int nLower, // нижняя граница
    int nPos // текущая позиция
);
```

6. Практическая часть

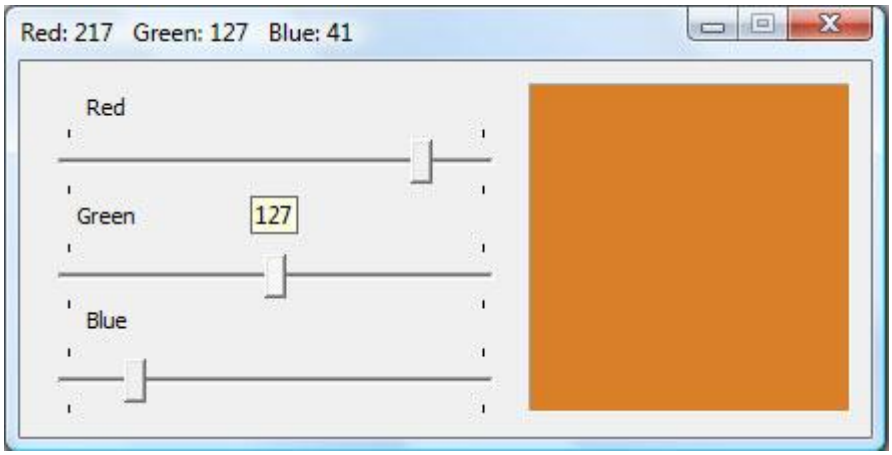
Дополнить приложение из домашнего задания предыдущего урока счётчиком, который предоставлял бы возможность выставить скорость игры в десятых долях секунды, что будет являться значением для таймера.

7. Подведение итогов

Подвести общие итоги занятия. Подчеркнуть основные способы создания регуляторов и счётчиков – с помощью средств IDE, а также посредством функции CreateWindowEx. Перечислить сообщения, наиболее часто используемые для управления регулятором и счётчиком. Акцентировать внимание слушателей на наиболее тонких моментах изложенной темы.

8. Домашнее задание

Разработать приложение, позволяющее с помощью трёх ползунков настраивать цвет фона индикатора.



Следует напомнить слушателям, что цвет фона индикатора устанавливается посылкой сообщения **PBM_SETBKCOLOR.** При этом в **wParam** передаётся 0, а в **lParam** – цвет фона (тип **COLORREF**), который можно задать с помощью макроса **RGB**.

```
#define RGB(r,g,b) \
((COLORREF)(((BYTE)(r)|((WORD)((BYTE)(g))<<8))|(((DWORD)(BYTE)(b))<<16)))
```

Например:

```
SendMessage(hProgress, PBM_SETBKCOLOR, 0, LPARAM(RGB(red,green,blue)));
```