Основи на входно/изходния интерфейс в среда Windows

1. Работа с текст

```
case WM_PAINT :
hdc=BeginPaint(hwnd, &ps);
графични обработки;
EndPaint(hwnd, &ps);
return 0;
```

```
hdc =GetDC(hwnd);
….
ReleaseDC(hwnd,hdc);
```

Ето един пример извличащ размера на текущата потребителска област: (обикновено в лог. координати. Зависи от mapping mode)

```
static short cxClient, cyClient;
....
        case WM_SIZE:
                cxClient = LOWORD (lParam);
                cyClient = HIWORD(lParam);
                ....
```

-Обикновено WM_SIZE се следва от WM_PAINT

-Управлението на scroll bar се поема от Windows: ефекти, съобщения, мишка

-Програмата следва да инициализира range на scroll bar; обработва съобщенията от bar; установява новото място на показалеца.

-Самият scroll се реализира с ScrollWindow(hwnd, xInc,yInc,..); Тя генерира WM_PAINT

особеност за .NET

за зла участ, ScrollWindow() е пропусната в Windows Forms (.NET)- има отделен

клас. Тогава следва наш код да инвалидизира част от кл. област и в OnPaint() да отново чрез код да се изобрази нужното

-Има и алтернатива – да се използва Win32 API функция

ОБРЪЩЕНИЯ КЪМ API ФУНКЦИИ ОТ MANAGED CODE

(напр ScrollWindow() е налична в User32.dll и си работи). Естествено, код с такова Повикване не е управлем и платформено независим.

BOOL ScrollWindow(HWND, int Xamount, int Yamount, CONST RECT *lpRect, CONST RECT lpCliprect)

Аргументите на повикването й не са пряко налични в .NET. Например:
-За HWND: класът Control има свойство Handle и конверсията от HWND към int е лесна;
- За последните 2 аргумента:
-структурата RECT, макар и дефинирана в Windows.h,
следва да се предефинираза да е със същите полета, (напр LONG сега е int (в C#))
-новодефинираната структура RECT да се предшества от атрибут [StructLayout…]
за да е допустимо извикването м/ду различните среди
- да се декларира като public static extern int функцията ScrollWindow(), преди повикването
- да се постави атрибут за местонахождението й: [DllImport('user32.dll')]

едва сега може да се повика старата API функция

Работа с текст при MFC подкрепа

CDC поддържа ред текстови ф-ии:

DrawText	**текст в правоъг. област**
TextOut	**ред на указани коорд.**
ExtTextOut	**ред със запълване на правоъгълника с фон и др.**
GetTextExtent	**изчислява ширината на низа в избрания шрифт**
GetTextMetrics	**връща шрифтовите размери**
SetTextAlign	
SetTextColor	
SetBkColor	

Шрифтът е GDI обект. В MFC той се представлява от CFont. Най-често се създава с CreateFont() (в pixels) или в пунктове:

```
CFont font;
font.CreatePointFont(12, _T("Times New Roman"));     // 12 пункта размер
```

след като създадете шрифт обекта следва да го изберете за употреба:

```
CPaintDC dc(this);
dc.SelectObjec( &font);
dc.SetBkMode(TRANSPARENT);
CString string = _T("Hello"));
CRect rect (….);
GetClientRect(&rect);
dc.SetTextColor( RGB(192,192,192));          //комбинира в COLORREF за GDI
dc.DrawText( string, &rect, ……);
```

CFont::CreateFontIndirect() за предварително създаден LOGFONT и за TrueType шрифтове (Times New Roman; Arial; Courier New; Symbol..)

Шрифтове

-**растерни** (съхранява се изображение – напр bitmap)
-**векторни** – дефинирани като набор линии и криви
-**TrueType** – контурни, позволяващи лесно мащабиране. Разработен от Apple и MS. В Windows има 13 базови такива (във файлове .ttf): Courier.., Times New Roman.., Arial.., Symbol.

-**OpenType** шрифтове – от 1997, подобрение на горния. Разработен от Adobe и MS и използван в езика за описание на страници – PostScript. В Windows 2000 са включени Georgia…, Palatino…, Tahoma…, Verdana…, и т.н.

(В диалога Fonts на Windows, шрифтовете TrueType (.ttf) се предхождат с TT файлове с шрифтове OpenType – икона с O)

-Windows Forms поддържа технология за изглаждане (чрез разместени форми, вкл. за LCD) за TrueType и OpenType шрифтове – свойство TextRenderingHint на клас Graphics. (Свойства на класа Graphics позволяват изглаждане и на линии и криви)

-шрифтовете се доставят в следните начертания: нормално, удебелено, наклонено, удебелен наклонен. Останалото се изработва

-размерът се указва най-често в пунктове (points). 1 пункт = 0.0138 инч = 1/72 инч. Подразбиращ се размер е 8 – 10 пункта за различните шрифтове. Размерът често се означава и в ‘em’ единици. Името идва от размера на квадратно парче метал за ‘M’ (използвано някога). Днес 1em е хоризонталния размер на знак ‘–’ за избрания шрифт.

Шрифове в .NET

-в клас Control (а следоват. и в наследниците му - а и във Form) има свойство Font, носещо информация за шрифта

- предвиден е клас Font и клас FontFamily (в System.Drawing) за работа с шрифт

напр: Font fontRegular = Font; //напр. Проперти Font на форма
Font fontBold = new Font(fontRegular, FontStyle.Bold);// нова версия

може и така: Font font = new Font("Times New Roman",24);

следва примерна програма (C#) - създава и визуализира (на екран и принтер)шрифтове Courier New, Arial, Times New Roman с 18 пункта във версии: нормална, удебелена, наклонена:

```
class FontNames: PrintableForm          //печата при клик в кл. област едновременно на екран и принтер
{       public new static void Main()
        {       Application.Run(new FontNames() );      }
        public FontNames()
        {       Text = "Font names";                    }
        protected override void DoPage (Graphics grfx, Color clr, int cx, int cy)
// потребит. ф-ия, която да прави гр. изход и се вика от реализациите ни на OnPaint() и PrintDocumentOnPrintPage() на Form
        { string[ ] astrFonts = {"courier New", "Arial", Times New Roman" };
        FontStyle[ ] afs = {FontStyle.Regular, FontStyle.Bold, FontStyle.Italic ..};
        Brush brush = new solidBrush( clr );
        float y = 0;
        foreach (string strFont in astrFonts)
        {   foreach (Fontstyle fs in afs)
                { Font font = new Font (strFont, 18, fs);
                  grfx.DrawString ( strFont, font, brush, 0, y); //'y' –настройва височината
                  y += font.GetHeight(grfx);        }}}     // на следващия ред според
}                                                         // шрифта
```

Мерни единици и взаимодействия

задаваме мерни единици на конструкторите на Font: мм, инч или простр координати. Реалното изобразяване се управлява и от 2 свойства на Graphics: PageUnit и PageScale. Съобразно техните настройки се изпълнява трансформацията на страницата (т.е. Тя се изпълнява само над членове на Graphics или ф-ии имащи арг. Graphics. Имаме:

(простр.координати) пространствена трансформация →(коорд.на страница)
трансформация на страница →(координати на устройство).

Как те си взаимодействат:

обектите Font са независими от наличието на обект Graphics (където са трансформациите);

1. взаимодействие между двата обекта (Font и Graphics) има само при методите:

1. **DrawString() – метод на Graphics, който има аргумент Font;**
2. **GetHeight() – метод на font, който има аргумент Graphics;**
3. **MeasureString() – метод на Graphics, който има аргумент Font.**

За тези случаи важат правилата:

а. пространствената трансформация (напр мащабиране с grfx.ScaleTransform(2,2), което удвоява) влияе и на графики и на текст по еднакъв начин.

б. за шрифтове, които са конструирани в метрични размери (напр. пункт, инч, мм) трансформацията на страницата не влияе на физическия размер на шрифта. Тази трансформация, обаче, влияе на координатите, подавани на DrawString(), както и на размерите, връщани от MeasureString() или GetHeight() (всички размери са в мм)

в. ако искаме размерът на шрифта да се изменя автоматично и да е в текущите единици на пространствените координати, то следва да го конструираме в единици: GraphicsUnit.Pixel или GraphicsUnit.World
(тогава шрифтът се изменя както и линии и области (според виканите методи))

2. Клавиатура

```
case  WM_KEYDOWN :
        switch (wParam)
        {       case VK_HOME:   //виртуален код на Home
                .....           ;       break;
                case  VK_END :  //виртуален код на END
                ... ;                   break;
```

```
case WM_KEYDOWN :
        switch (wParam)
        {
        case VK_xxxx:
        SendMessage(hwnd, message,wParam,lParam);
        break;
        .......
```

WM_KEYDOWN	виртуален код на a
WM_CHAR	ASCII код на a
WM_KEYUP	виртуален код на a

Тогава, фрагментът обработващ клавиатура може да се промени по следния начин:

```
case  WM_CHAR :
        switch(wParam):
        {       case '\b':      // backspace
                .....;          break;
                case 'n' :      // буква n
                .......;        break;
                default:
        ..........
```

Клавиатура в 32 битова среда (допълнения)
Съобщенията се насочват към фокусирания прозорец
Промяна на фокус: WM_SETFOCUS WM_KILLFOCUS
и съответно:
ON_WM_SETFOCUSS ON_WM_KILLFOCUSS в картите на съобщения

```
void CMainWindow::OnSetFocus( CWnd* pOldWnd)
{
        .........
}
```

Прехвърляне на фокус:
CWnd::SetFocus();
CWnd* pFocusWnd = CWnd::GetFocus()

клавиатурни особености в .NET среда

-библиотеките на Windows Forms обработват много от клавиатурните събития. Напр. :
от меню, вкл. клавиатурни комбинации CTRL+..
от диалогови прозорци, вкл. и навигация между контроли с Tab..

-класът, който имплементира обработки от клавиши, следва да е наследник на Control –
напр. Form и да има 'input focus'. Това е 'активната форма'

-с активната форма са свързани:
статично свойство Form.AcriveForm – връща активна форма (само ако
тя е създадена от вашето приложение)
void Activate() - метод на Form
Activated/Deactivated - събития на Form (OnActivated()..)

-активната форма и входният фокус са различни неща – ако формата съдържа контроли

-класът Control има функционалност за:
- анализ на събития от клавиатура (KeyDown..) и свързани методи;
- попълнен от събитието обект KeyEventArgs с много свойства;
- изброяване Keys (най-голямото в .NET) идентифициращо всички клавиши;
-пример за проверка за натиснат 'x':

```
protected override void OnKeyDown(KeyEventArgs kea)
    {
    if( kea.KeaCode == Keys.x)
    ....
    }
```

или пък:

```
    if( kea.KeyCode == (Keys) (int)'x')
```

Каретка – текстов курсор (локален ресурс в рамките на нишка)

Създава се при фокусиране на прозорец и се показва в OnPaint() от ::BeginPaint() /::EndPaint(). Местенето й е от програмиста.

CWnd има 7 ф-ии за работа с каретка:

CreateCaret **създава я от битмап**
CreateSolidCaret()
GetCaretPos
SetCaretPos
ShowCaret
HideCaret и API ф-ия: ::DestroyCaret()

Как е с каретката в .NET

- в контроли от тип TextBox и др., код в контрола отговаря за поддръжката на текстовия курсор
- за зла участ архитектите на Windows Forms са пропуснали класове, които да я поддържат
  Липсва напр. клас Caret.
  Отново може да се прибегне към обръщения до старите Windows API функции :

```
[DLLImport('user32.dll')]
public static extern int CreateCaret(..)
[DLLImport('user32.dll')]
public static extern int DestroyCaret(..)
[DLLImport('user32.dll')]
public static extern int ShowCaret(..)
[DLLImport('user32.dll')]
public static extern int HideCaret(..)
```

и те да се обвият в потребителски клас в .NET среда - Caret напр. в собствен namespace (естествено създава се unmanaged code фрагмент)
Създава се напр с: Caret caret = new Caret(form) и т.н.
В този клас тряба да има и манипулатори на събития, напр OnGetFocus(), OnLostFocus()

Мишка

-съобщения към текущия прозорец.

- към опашка на thread, който ще ги обработва

- при движение на мишката WM_MOUSEMOVE. В некл. област: WM_NCLBUTTONDOWN

- при натиснат бутон : WM_LBUTTONDOWN
WM_LBUTTONUP
WM_LBUTTONDBLCLK

-съпътстващи парам. – местоположението спрямо горе/ляво в пиксели (CDC::DPtoLP()),
Shift.Ctrl,област..

- Как да определим кога курсор влиза и излиза от рамката на прозорец:

влизането : в опашка се отлага WM_MOUSEMOVE.

интересна ф-ия ::TrackMouseEvent() (нужна настройка напр:#define _*WIN32*_WINNT 0x0400)

С нея можем да укажем приложението (с параметър HWND) получаващо съобщния и:
WM_MOUSELEAVE и WM_MOUSEHOVER

(няма асоциирани с тях ф-ии в MFC→ ON_MESSAGE…)

Прихващане на мишка:

CWnd::SetCapture()

::ReleaseCapture() ***CScrollView::OnMouseWheel()***

курсор на мишка

1.Създаване на курсор чрез WNDCLASS :

CString strWndClass = AfxRegisterWndClass(…,AfxGetApp()

LoadStandartCursor(IDC_CROSS),……..); **//в констр. на CMainWindow**

……..

Create(strWndClass,…….); **// в CFrameWnd::Create()**

2. С API ф-ия ::SetCursor(m_hCursor); //при WM_SETCURSOR напр., което се подава при движение на мишката

3. CWaitCursor wait; // създава пясъчен часовник до срещане на wait.Restore();

Мишка в .NET – особености

класовете от Windows Forms се грижат за мишката, ефектите и входа от нея.
Напр. Скролиране, когто има лента за скролиране
- контролите също капсулират интерфейса от ниско ниво за клавиатура и мишка и осигуряват такъв от високо ниво за програмиста;
-за специфични приложения:

клас SystemInformation дава подробна информация за мишка (бутони, wheel.._
клас Control дефинира 9 събития с мишка и съответни методи, наследени от всеки произлизащ от Control клас
клас MouseEventArgs (подаван като арг. на методите), в свои свойства дава допълваща събитието информация

в .NET

-натискане на бутон върху контрол или кл. област на формата, ‘прихваща мишката’ и кара всички последващи събития от нея да се изпращат към тях. До отпускане на бутона.
Това може да се промени от свойство Capture на клас Control.

-курсор на мишка

той е обект от тип Cursor, дефиниран в System.Windows.Forms
предефинирани поведения на курсор са достъпни от класа Cursors на същото пространство имена. Той се състои само от 28 описани форми на курсора. Напр: Arrow, Cross, Hand, Help, WaitCursor ..

```
Cursor.Current = Cursors. WaitCursor;
```

// свойството Current, както и Position и Clip са статични свойства на Cursor класа

Задава правоъгълна област в която
са ограничени движенията на мишката

Представлява визията на текущия обект - курсор

4. Таймер (до 2^64 ms)

Ето един фрагмент от програма, дефиниращ 2 таймера:

```
#define    TIMER_SEC          1
#define    TIMER_MIN          2
.....
SetTimer(hwnd, TIMER_SEC, 1000, NULL);
SetTimer(hwnd, TIMER_MIN, 60000,NULL);
....
            case WM_TIMER :
switch(wParam)                  //съдържа идентификатор на таймер
{case TIMER_SEC :               // обработка веднъж в секунда;
                    break;
case TIMER_MIN :                // обработка веднъж в минута;
                    break;
```

Идея за CALL-BACK функция

```
WORD FAR PASCAL TimerProc(HWND hwnd, WORD message,
                                        WORD wParam,LONG lParam)
{//          отработване на съобщенията от таймера;     return 0;  }
EXPORT              xxxxx
                    TimerProc
```

```
FARPROC             lpfnTimerProc;      // декларация на дълъг указател
......
lpfnTimerProc = MakeProcInstance (TimerProc, hInstance);
.....
SetTimer(hwnd,1, Interval, lpfnTimerProc);
```

CWnd:: SetTimer()
CWnd::KillTimer();

MFC макросът ON_WM_TIMER насочва WM_TIMER към
CMainWindows::OnTimer(UINT nTimerID);

1.Приложението с таймер (чрез прихващане на съобщението) би следвало да изглежда така:

```
BEGIN_MESSAGE_MAP ( CMainwindow, CFrameWindow)
ON_WM_CREATE()
ON_WM_TIMER()
END_MESSAGE_MAP
int CMainWindow::OnCreate(.....)
{.....if(!SetTimer(ID_TIMER_XX,100,NULL)) {..}
return 0;}
void CMainWindow::OnTimer(UINT nTimerID)
{          // например изрисува нещо през 100 мс    }
```

2.Ако искаме да свържем таймера с callback ф-ия:

(не са по-бързи от използване на съобщения. Предимство е че не се създава запис в картата на съобщенията, а само флаг. DispatchMessage()насочва към callback ф-ия или картата на съобщенията)

```
SetTimer( ID_TIMER, 100, TimerProc);
.....
void CALLBACK TimerProc(HWND hWnd, UINT nMsg, UINT nTimerID, DWORD dwTime )
{ .....}
```

*обработка без WM_TIMER през ::DispatchMessage();

•Не получава lParam параметър в който да се подаде this - той служи за достъп до нестатичните методи на класа

• Може CMainWindow* pMain = (CMainWindow*)AfxGetMainWnd();

•*често служат за разширяване стандартното действие на API ф-ии.

•Callback ф-ията е глобална или статична за да не получава this за класа си

•Позволява да й подаваме параметри (или ОС да изработва такива)

3.Определяне на времето

(напр във ф-ия OnTimer(UINT nTimerID..) {…})

CTime е клас за време и дата.

```
CTime time = CTime::GetCurrentTime();
int second = time.GetSecond();
int nMinute = time.GetMinute();
int nHour = time.GetHour() % 12
```

Таймер в .NET

-дефинирани са 3 класа с името Timer, в: System.Timers; System.Threading; Sytem.Windows.Forms
-в Windows 2000 дава сигнал на 1 ms, но закръглено към следващото, кратно на 10
-елементи на вграждане в програма:

```
Timer timer = new Timer();
....
OnTick(EventArgs ea)
```

събитие

```
или потребителски:
void TimerOnClick(object obj, EventArgs ea) {.....      }
......
timer.Tick += new EventHandler(TimerOnTick);
.....
timer.Interval = 200;
......
timer.Enabled = true;
```

Има 2 свойства get/set

-структурата DateTime в пространството System служи за инициализация. Има 15 свойства за Четене (hour, minute,second..). Свойството Now я преинициализира с текущото време:

```
DateTime dt = DateTime.Now;
```

-време от DateTime се преобразува с нейни ф-ии (ToLocalTime()..) към локално, UTC или Гринуич

-вътрешно времето в DataTime е в тактове (Ticks – 100 ns) спрямо полунощ, 01.01.0001 като long, 64 р-дно число и методите го преобразуват към дата и час.
-обектът TimeSpan е подобен на DateTime, но съхранява броя тактове спрямо 2001 г.
-конструкторите на DateTime имат параметър за тип календар. Има 7 подкласа на клас Calendar за различни типове календари: Грегориански – подразбиращ се, Корейски, Японски, Юлиянски, Ислямски..

5. Контроли (прозорец – винаги дъщерен) за извеждане/въвеждане на инф.

*** определяне на родител: hwndParent = GetParent(hwndChild);**

*** съобщения към родител:SendMessage(hwndParent,message,wParam,lParam);**

* Изпращат най-често WM_COMMAND с wParam и lParam

* MFC асоциира методи на обвиващите класове с нотификационните съобщения от контролите

Бутони (pushbutton, check box, radio, group box). Обвиващ клас CButton

- **Базират се на структурата "BUTTON" от тип WNDCLASS;**
- **Тият се указва при създаване с параметър за стил: BS_PUSHBUTTON, …**
- **CreateWindow("button",текст, стил, x, y, x-размер, y-размер,hwndparent..)**

  Разрушават се с родителя

  Изпращат:

  WM_COMMAND

*** Пример за създаване на бутон в MFC среда:**

```
CButton m_wndPushButton;
m_wndPushButton.Create(_T("Start"), WS_CHILD | WS_VISIBLE |
                       BS_PUSHBUTTON, rect, this, IDC_BUTTON);
```

- **SendMessage(hwndButton, BM_SETSTATE,1,0); // симулира натиск.**
- Свързване с нотификационно съобщение:

  ON_BN_CLICKED (IDC_BUTTON, OnButtonClicked)……

  Void CMainWindow::OnButtonClicked() { …}

- **Полета за отметка: BS_CHECKBOX; BS_AUTOCHECKBOX; BS_3STATE**

  m_button.SetCheck(BST_CHECKED);….

  void CMainWindow::OnCheckBoxClicked() {..}
- **Радио бутони: BS_RADIOBUTTON или BS_AUTORADIOBUTTON**

  *** може да се формира група m_button.Create(…,WS_GROUP,..)…**

*** Group Box контроли – само за визуално разделяне**

Контрол за разлистване (listbox). Обвиващ клас CListBox.

•WM_COMMAND, нотификационни съобщения (ON_LBN…) и макроси за асоцииране; скролиране при WS_VSCROLL…; поддържа интерфейс към клав.

CListBox m_wndList;

m_wndList.Create(WS_..|.., rect, this, IDC_..);

•Стандартно list box контрола се прерисува автомат. След всяко добавяне/изтриване на елемент

•Контрол с много колони : LBS_MULTICOLUMN;

•m_wndList.InsertString(..);

.DeleteString(..), GetCurSel(), FindString(), GetString()…

•m_list.SelectString(.., _T(“Nakov”));

Контрол за редактиране (edit) . Обвиващ клас CEdit.

- CreateWindow(“edit”, …);
- бутонът изпраща WM_COMMAND с нотификационен код (в lParam).
- Пример за многоредов edit контрол е Notepad
- Клас CEdit (до 60KB); (друг клас е CRichEditCtrl)

```
CEdit::SetMargins()          // задава размерите на полетата вляво и дясно
CEdit::SetRect()             // задава редакторската област в контрола
CEdit::SetLimitText()        // задава макс. брой символи  (подр. се 30000.)
CEdit::SetWindowText()       // вмъква текст
CEdit::GetWindowText()       // извлича текст (ф-иите са наследени от CWnd)
        ::Clear()
        ::Cut()              за селектиран текст с взаимодействие с clipboard
        ::Copy()
        ::Paste()
```

* изпраща нотификационни съобщения (промяна фокус, скролиране в контрола, запълване с макс. брой символи..)и асоциирани с тях ф-ии:

ON_BN_CLICKED (IDC_BUTTON, OnButtonClicked)

Статичен контрол . Обвиващ клас CStatic

Бива: 1. Само текст
2. Правоъгълник
3. Изображение (от битмап, икона, курсор..)

* По подразбиране не изпраща нотификационни съобщения към родителя. Ако е създаден с SS_NOTIFY нотифицира при фокус, кликване и др.

Combo box контрол. Обвиващ клас CComboBox

- Обединява едноредов edit с list box
- Бива:
  - 1. Прост — списъкът е постоянно изобразен.Позволява и директно въвеждане на текст
  - 2. Drop-down — списък – само при поискване
  - 3 drop-down list — както 2, но не позволява директно въвеждане на текст
- Rect параметърът при създаване да е достатъчен за поемане edit + list частите
- Изпраща над 20 нотификации

CScrollBar

- Позволява позициониране навсякъде
- за разлика от останалите класически контр. изпраща WM_VSCROLL и WM_HSCROLL асоциирани с методи на класа
- има: GetScrollPos(), SetScrollPos(), SetScrollRange() и др.

РАЗШИРЯВАНЕ ФУНКЦИОНАЛНОСТТА НА КОНТРОЛИ

Пример: edit контрол, който приема само числови стойности:

```
class CNumEdit : public CEdit
{
  protected:
  afx_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);
  DECLARE_MESSAGE_MAP()
};

BEGIN_MESSAGE_MAP(CNumEdit, CEdit)
ON_WM_CHAR()
END_MESSAGE_MAP()

void CNumEdit::OnChar((UINT nChar, UINT nRepCnt, UINT nFlags)
{
  if((( nChar >= _T('0')) && (nChar <= _T('9'))))
      CEdit::OnChar(nChar, nRepCnt, nFlags);
}
```

Графични бутони: в MFC има 3 производни класа:

- CCheckListBox // list box с отметки (SetCheck()..)
- CDragListBox // list box, поддържащ drag&drop
- CBitmapButton // позволява вместо текст -- картинка

Общи контроли (над 20, MFC осигурява класове за тях)

•WM_NOTIFY вместо WM_COMMAND
- в lParam има указател към структура с данни за събитието и в wParam – IDC_
- IE разширява броя нотификационни съобщения
- класове:

CAnimateCtrl; CComboBoxEx; CDateTimeCtrl; CIPAddressCtrl; CListCtrl; CProgressCtrl,CMonthCalCtrl; CRichEdtCtrl; CSliderCtrl; CSpinButtonCtrl; CStatusBarCtrl; CTabCtrl..

Създаване:

#include <afxcmn.h> *//съдържа декларациите на контролите*

- *CProgressCtrl wndProgress;*
  ndProgress.Create(стил, размери в правоъгълник, this, IDC_…);
- *Другият начин на създаване е чрез вграждане в диалогов прозорец с граф. редактор.*

Trackbar контрол. Обвиващ клас CSliderCtrl

- *Позволява управление от мишка и клавиатура*
- *Има настройки за изобразяване, за позиции, за дискрети нарастване*

Spin контрол (up-down контрол). Обвиващ клас CSpinButtonCtrl

- *Интерфйс към мишка и клавиатура*
- *Задаване диапазон, дискрет, граници*
- *Асоцииране към buddy контрол (метод SetBuddy()), който най-често е едноредов edit*
- *Изпраща WM_NOTIFY към родителя*

Списъци с изображения -Image List . Обвиващ клас CImageList

(използват се в tree view, list view, combo box extended)

•Създаване

```
CImageList il;
il.Create(IDB_BITMAP, 18, …);   …
```

ID на bitmap ресурс с ‘n’ изображения, 18 пиксела високи

•Позволява указване на ефекти (напр. маска) за изрисуване на картинките впоследствие

• изчертаване на изображение от списъка:

```
il.Draw(pDC, индекс, нач.координати, ефекти при чертане);
```

Контрол с включени изображения: ComboBoxEx: public CComboBox

**** CComboBoxEx::SetImageList(& обект от тип CImageList)***

•CComboBoxEx::InsertItem() *//добавя елемент към контрола с указан текст,*

//асоциирано съобщение,изображения, атрибути

:: DeleteItem()

::SetItem() *//модифицира*

•Позволява изобразяване в различни стилове (напр. с отстъп)

• бива: прост, падащ и с падащ списък (както CComboBox)

ToolTip контрол – прозорец с помощен текст. Обвиващ клас CToolTipCtrl

- появява се динамично при задържане на мишка върху бутон, toolbar или контрол
- асоциира се с “инструмент” – друг прозорец, контрол или правоъгълна област.
- Задава му се текст

Контроли на прогрес . Обвиващ клас CProgressCtrl

```
ProgressCtrl ::SetRange();   //задава диапазон от.. до..
             ::GetRange();
             ::SetPos();        // задава текуща позиция
```

пример:

```
m_wndProgress.SetRange(0, 100);
m_wndProgress.SetPos(0);
for( int I =0; I < 100; I++) {
        m_wndProgress.SetPos(I); ::Sleep(25);
}
```

Контрол за анимация . Обвиващ клас CAnimateCtrl
*** Поема avi филм (примери: Filecopy.avi, FindFile.avi)**
CAnimateCtrl::Open() // зарежда AVI файл
CAnimate::Play(); //стартира анимациятаи връща упр. Изисква
// начален, краен фрейми и брой итерации

Контрол календар CMonthCalCtrl

ето пример за задаване текущата дата:
m_wndCal.SetCurSel(CTime(2008, 3, 25, …);

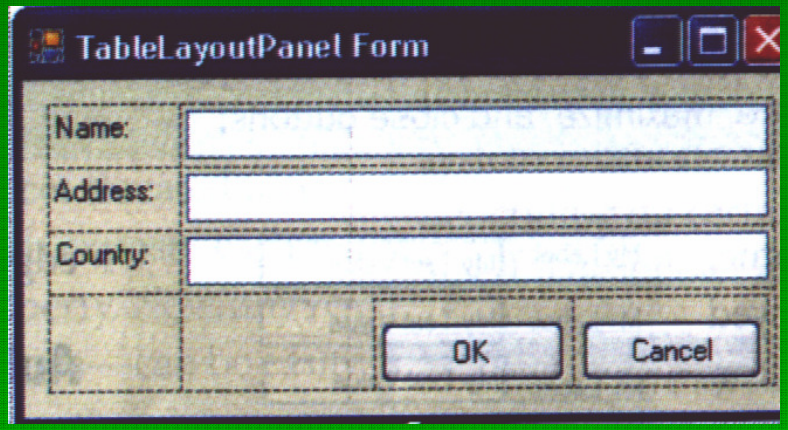
Контрол за избор на дата и час CDateTimeCtrl (Date-time picker DTP)
Като edit ctrl, но за дата, може и с визуализиране на календара
дата: 9/03/02 или Thursday, September 25,2001
Час: HH:MM:SS AM/PM

Контрол за IP адрес. Обвиващ клас CIPAddressCtrl

- Работи с 4, 8 битови целочислени стойности
- Методи SetAddress(…)
  GetAddress(),
  SetFieldRange() (0 – 255)

Някои нови контроли и компоненти (.NET Framework 2.0)

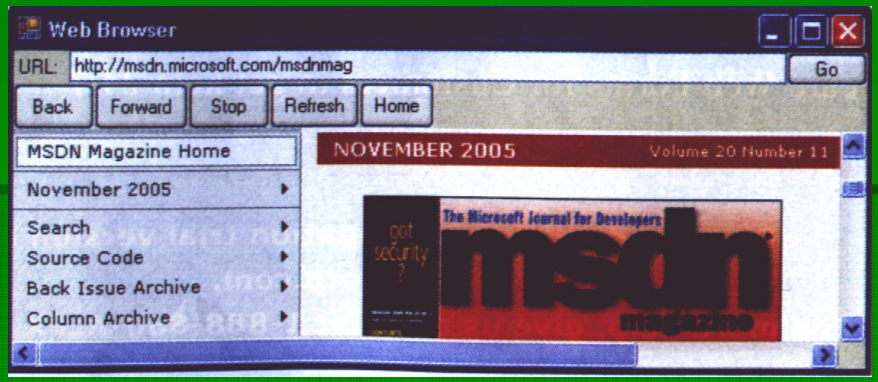
- **TableLayoutPanel контрол:** lets you define cells and rows for laying out controls in a fixed grid.



- - позволява дефиниране броя редове и колони в следните стилове:
  - 1. AutoSize- автоматично resizing според съдържанието;
  - 2. Percent - размер в зададен процент от височина или ширина;
  - 3. Absolute – в pixels;

  Във всяка клетка може да се позиционира друг контрол (напр. Бутон, както по-горе)

- **Разширен WebControl - Brouser контрол**



пример за навигация в web пространство с този контрол:

```
void goButon_Click(object Sender, EventArgs e) {
        this.webBrouser.Navigate(this.urlTextBox.Text);
        }
```

// преходът към нова страница е съпроводен със запалване на Navigated event преди
// да се извърши download. Излъчва се и ProgressChanged event за проследяване.

Пр. за други действия:

```
this. webBrouser.GoBack();              // към предишно посетена страница
this.webBrouser.Stop();
this.webBrouser.Refresh();
this.webBrouser.GoHome();
```

BackgroundWorker компонент

- За изпълняване на продължителни операции в фонов режим. Напр. търсене. Един изход е пускане на ‘worker thread’ за асинхронно изпълнение на такива операции.

- BackgroundWorker капсулира сложните задачи по менажиране на работна нишка. Включването му във функционалността на приложението става с просто ‘влачене’ на компонента във формата.
  BackgroundWorker.RunWorkerAsync() //стартира

- Събитие DoWork се прихваща за описване функционалността на процеса
- Контролът позволява thread-safe предаване на данни между UI и worker threads; поддържа ‘progress reporting’, прекратяване на нишки и др.

Пример:

```
Public class TaskForm : Form
{
    void StartOperationButton_Click(..)
    { // create data to pass to worker thread
    MyData data = new MayData();
....
    this.backgroundWorker.RunWorkerAsync(data);          }          //стартиране

// в нишката
    Void backgroundWorker_DoWork (object sender, DoWorkEventArgs e)
        {  MyData data = (MyData) e.Argument;
        // execute long-running asynchronous operation
        ..}}
```

- **DataBinding fundamentials**

1. BindingSource компонет – за свързване към източници на данни с богати възможности за управление.

Добавянето става в развойната среда :

Data Source Configuration Wizard (Data| Add New Data Source)

-Източник на данни може да е : база данни, Web услуга, обект;

- **При източник БД – добавя DataSet обект (както е показано на картинката)**
- **Позволява редактиране, преконфигуриране или refresh над DataSet**
- **Контроли като DataGridView когато се вкарат (droped) във форма, автоматично питат за data source.**

- **BindingSource**

Изобщо, свързване с източник на данни обикновено става през DataSet (за БД от релационен тип);

Свързване може да има и с : таблици, XML данни, .NET обекти. Всички те могат да се привързват като източници към форма. Това става с определени усилия: напр реализация на IBindingList интерфейса.

- Компонентът BindingSource автоматизира това за всеки тип източник на данни и имплементира горния интерфейс вместо програмиста.
- За да се конфигурира свързване към някакъв тип източник на данни:
  - 1. drop на DataSource компонент във формата
  - 2. подходяща настройка на properties: DataSource и DataMember. Ето пример за свързване с DataSet:

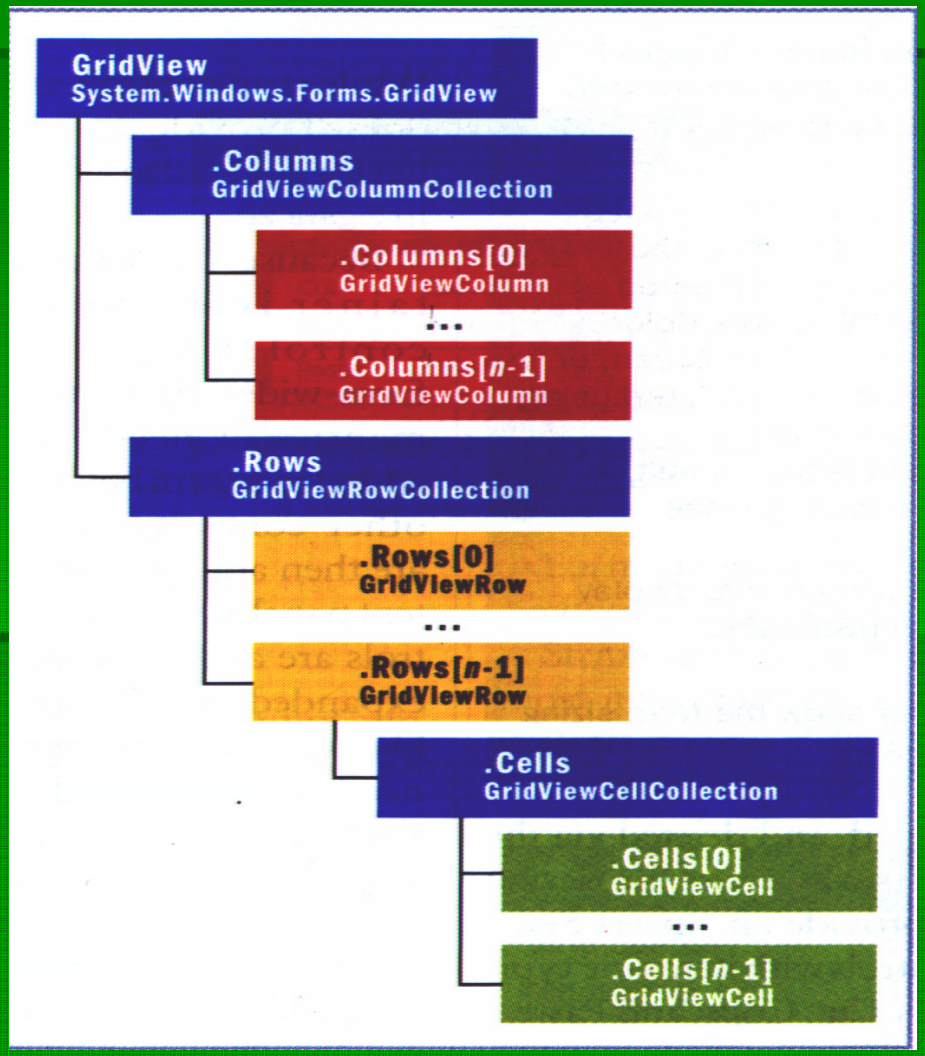
```
Public class DataBindingForm : Form        {
    Public DataBindingForm()    {
        ……
    this.BindingSource.DataSource = this.northwindDataSet;          // БД
    this.bindingSource.DataMember = “Employees”;                    //таблица
    }
    }
```

Оттук нататък източника на данни за тази форма и за вградените контроли е дефиниран

- **DataGridView** – обогатен grid контрол

(старият DataGrid още съществува – за навигация и редактиране на йерархични данни)

Нов е обогатеният обектен модел на GridView:



- **DataGridView позволява:**
- Лесна настройка на стилове, формати;
- Изобразяване на различни типове данни, вкл. картинки;
- Мощни възможности за динамично преконфигуриране (напр. пренареждане на колони)
- Повече от 100 привързани с него events за navigation, editing, validation, error handling и т.н.

Запълване на DataGridView с данни обикновено става след привързване към data source (с настройка на property DataSource: