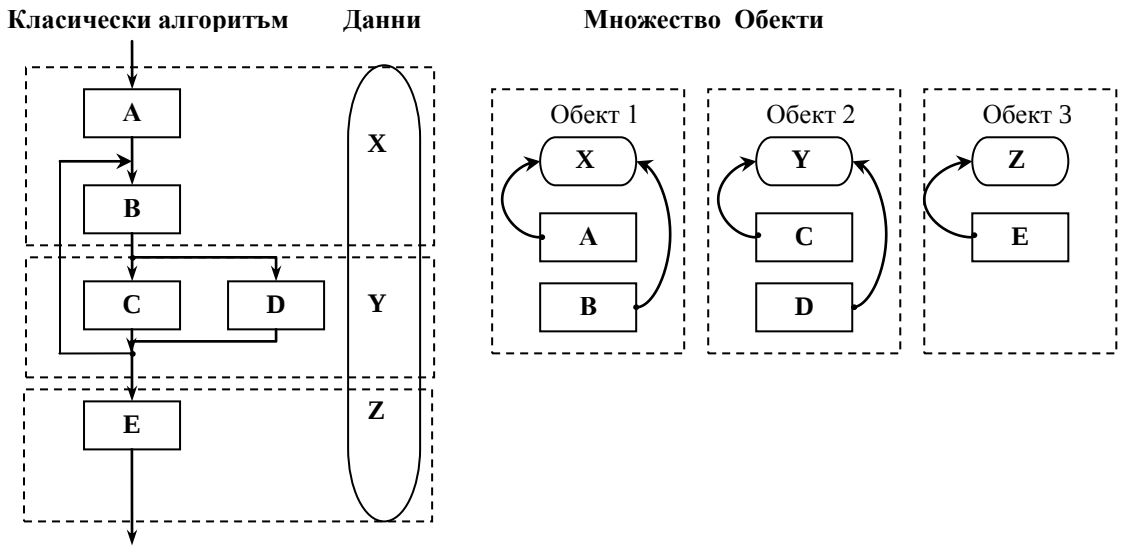
Лекция 7

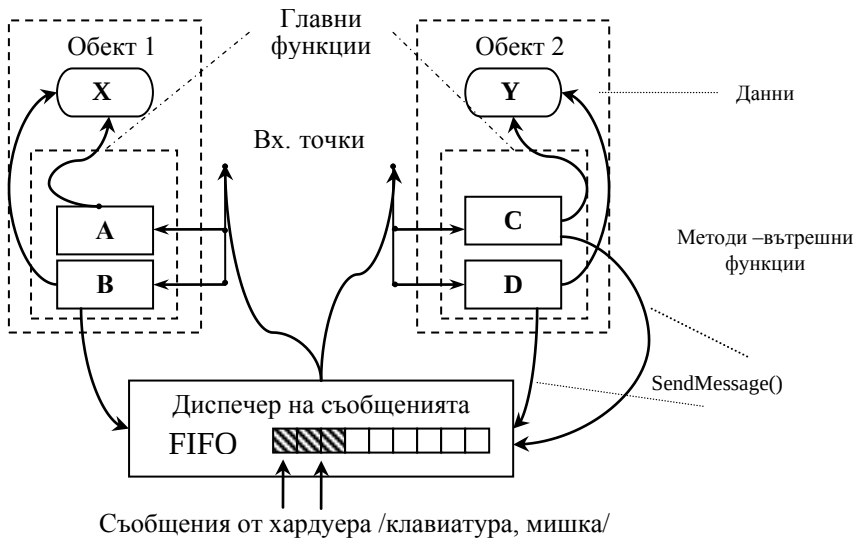
Обекти, полиморфия и наследяване на обекти в средата на Windows

Обектно ориентираното програмиране е основна концепция в Windows.

Същност и модел на обектите



Фиг. 14. Обектен модел на организация на данните



Фиг. 15. Обмен на съобщения между обектите

Фиг. 14. позволява да се сравнят последователния и обектния модел на обработка на данните. Обектния модел организира данните и методи за обработката им в понятието „обект” . С буквите „A”, „B”, „C”, „D”, „E” са

означени програмни функции, а с „X”, „Y”, „Z” структури от данни. Обект 1 включва структура от данни X и програмните функции A и B. В лявата част на фигурата е показано структурирането на обектите в рамката на класически алгоритъм, а в дясната част структурирането по данни. Данните от всеки обект са достъпни само чрез функциите (методите) на този обект. Този механизъм се нарича „***капсуловане***” (затваряне) на обектите.

Съобщенията между обектите се осъществяват като:

- Изпълняват се командите към обектите, които съдържат данни;
- Командите определят последователността на изпълнение на функциите;
- Обектите обменят данни само чрез съобщения.

Механизмът на обмен на съобщения между обектите за разглеждания по-горе пример е представен със схемата от Фиг. 15.

Обекти в Windows могат да съдържат:

- Структура от данни, която представя описание на обекта като: координати, цвят, тип, размер на прозорец и други описателни атрибути);
- Главна функция която обслужва съобщенията постъпващи към обекта.
    - Функцията се прикача към обекта, когато той се инициализира (възниква);
    - Всички функции имат общ заглавен оператор стандартни параметри и кодове на съобщенията;
    - Структурата на функцията е от тип: switch(разклонител) по кода на съобщението;
    - Функцията анализира всяко съобщение и реагира на някои от тях (промяна на размера, съдържанието или атрибутите на прозорец)
- Подчинени функции, към които се обръща главната функция (функции на приложението);
- Стандартни функции на Wndows, които се използват от главната или подчинените функции (TextOut, BitBlt,….)

.

<u>Обектно програмиране в средата на Windows</u>

Основните понятия и принципи на обектно ориентираното програмиране (ООП) са : ***обект, клас, капсулиране (затваряне), наследяване и полиморфизъм***.

Класът се явява обобщение на понятието за тип на данните и задава свойства и поведение на обектите на класа , наричани още и екземпляри на класа. Всеки обект принадлежи на някои клас. Класът може да се представи като обединение на данни и процедури, предназначени за техните обработки. Данните на класа се наричат също така и ***променливи на класа***, а процедурите –***методи*** на класа. Всеки клас съдържа данни и методи.

Капсулирането (затваряне) на класовете е обработката на данните само чрез методите на класа. В C++ променливите на класа се наричат ***данни-членове*** на класа (data member), а методите – ***функция-член*** на класа (member function). Променливите определят свойствата или състоянието на обекта. Методите определят поведението на обекта.

Наследяване на свойства и поведение от родителския клас. Нека е определен клас A, тогава може да се определи нов клас B, наследяващ свойствата и поведението на обектите от клас A. Това означава, че в класа B ще бъдат определени променливи и методи от клас A. Класът B се явява породен от клас A. Класът A се явява родителски (базов) по отношение на B. В производен клас може да се зададат нови свойства и поведение, определяйки нови променливи и методи. Може да се ***пре-определят*** методите на базовия клас. ***Пре-определянето*** (overloading) на методите на клас A – това е определяне в клас B на метод с име, вече появило се като име на някой метод от клас A. Методът на базовия клас може да се обяви като ***виртуален*** (с атрибут virtual). Тогава при пре-определянето на метода в производен клас трябва да се съхранява броя на параметрите на метода и техните типове. Виртуалността обезпечава възможност за написване на ***полиморфни функции***, чиито аргументи могат да имат различни типове (класове) при различни обръщения към функциите и при това ще изпълняват действия, специфични за типа на фактически предадения аргумент. Например, нека в клас A е определен виртуален метод VirtualMethod(), който издава съобщението „Аз съм метод от клас A”. Нека да пре-определим метода в класа, така че да издава съобщение „ Аз съм метод-член на клас B”. Тогава следва пре - определяне от вида:

```
class A   // Определяне на клас A
{
```

```
        public:
// Виртуален метод на класа A
virtua void VirtualMethod() {Издава съобщение „Аз съм метод на клас A”}
}

// Определяне на клас B
class B: public A
{       public:
// Виртуален метод на класа B
virtual void VirtualMethod() {Издава съобщение „Аз съм метод на клас B”}
}

// Полиморфна функция. !!!! В качеството на аргумент тя приема указател
// към клас A, които е базов за B
void ShowMessage(A* x)
{
// Извикване на метода на обекта  VirtualMethod() x
x->VirtualMethod();
}
// Създаване на ОбектA на класа A
A ObjectA;
// Създаване на ОбектB на класа B
B ObjectB;
// Извикваме полиморфна функция с аргумент – указател към обект от клас A
// Ще се изведе съобщението: „Аз съм метход на клас A”
ShowMessage(&ObjectA)
// Извикваме полиморфна функция с аргумент – указател към обект от клас B
// Ще се изведе съобщението: „Аз съм метход на клас B”
ShowMessage(&ObjectB)
```

Класът B може да е базов за следващ клас и.т.н. Всички породени класове се наричат ***наследници***, а класовете от които са породени се наричат предци или ***родители***.

Определянето на класовете (интерфейс на класовете) се намира в заглавните файлове ***.h,** а реализацията на функциите на класовете се намира в едноименния файл с разширение ***.cpp.** Тази организация структурира

изходния текст, облекчава управлението на данните и позволява да се раздели компилацията на модулите на програмата.

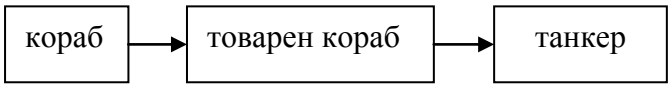
Наследяване на свойства в Windows

Наследяването е свойство на обектите подредени в йерархия, което позволява чрез йерархия от класове, общите свойства да се изведат на по-високо ниво в йерархията. В Операционните системи наследяването има динамичен характер.

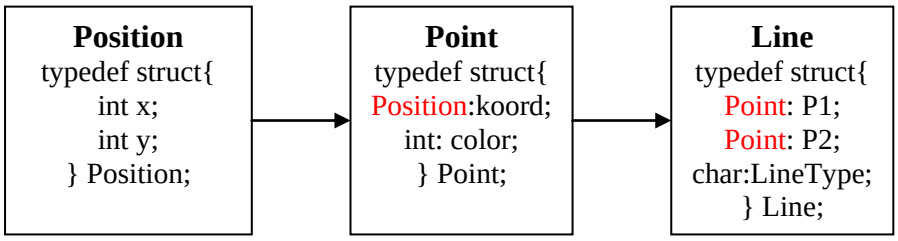
Йерархичната структура на обектите се характеризира с подреждане на обектите в нива така че всеки обект да има 1 предшественик, ако не е корен и да има наследници, ако не е листо. ***Основният принцип на наследяването*** е, че всички наследници наследяват данните и обслужващите функции на предшественика си. Всички наследници се явяват ***инстанции*** (***конкретизации***) на предшественика. Всички предшественици се явяват обобщения на наследниците си. ***Предшествениците са*** чисто абстрактни класове и съдържат празни методи. Методите на наследника могат да припокриват методите на предшественика. ***Полиморфията е свойство на обектите да реализират определена абстрактна функционалност по свой специфичен начин и налага използването на стандартизирани методи.*** Напр. Инстанциране и изтриване на обекти. **……**Ref=new_class_Name води до инстанциране в ***heap*** (таблица със входните точки - стартовите адреси на методите).

Следват примери на йерархично изградени обектни структури:

Пример: 1



Пример 2



Следващия пример представя наследяването в Windows чрез регистрация на класове при създаване на прозорци, което става по време на изпълнението на приложението.

Register Class (...) //обект със свойства за определен тип прозорци)

……..

CreateWindows (…..) - //собствени свойства

Полиморфия в Windows

Полиморфията е свойство на обектите да реализират определена абстрактна функционалност по свой специфичен начин. Тези функции са полиморфни на абстрактната функционалност и представляват нейните конкретизации. Това означава:

- Стандартизиране на управлението на множество обекти;
- Въвеждане на номенклатура на методите;
- Унификация на обектите.

Полиморфията се постига чрез препокриване на абстрактни обекти и методи.

Полиморфия при методите

Нека като абстрактна функционалност е използван абстрактен метод като Move(Id,x,y), който премества визуален обект в прозорец на нова позиция с нови координати. Съответните му полиморфни функции (методи), които биха могли да го препокрият са: преместване на векторен обект (линия, квадрат); преместване на растерен обект – bitmap; снимка или преместване на текстов обект –низ.

Полиморфизмът се реализира чрез програмиране на специфични (подчинени) функции, които се активизират от различните главни функции на обектите за обработка на едно и също съобщение: Напр. Съобщението WM_PAINT предизвиква пречертаването на всички прозорци, към които се изпраща, а върху различните прозорци може да са изобразени различни графични или текстови обекти. Такъв е случаят когато се промени размерът на някои от прозорците на екрана.

Подходящ пример за пре-определяне на методите на класовете е приложение където функцията за изобразяване на векторен обект се пре-определя по два различни начина.

В базовия клас „Shape" функцията-метод е определена само като (virtual void Draw(HDC hdc)) В двата класа наследници (class Line, class Rect) тя се преопределя по два различни начин, съответно да изобрази линия или

правоъгълник.

```
enum shpType
{
        line, rect
};
//
class Shape
{
public:
        POINTS begin;
        POINTS end;

        Shape() {};
        ~Shape() {};

        Shape(const Shape& s)
   {
      begin = s.begin;
      end = s.end;
   }

   Shape& operator=(const Shape& s)
   {
      begin = s.begin;
      end = s.end;
      return *this;
   }

        virtual void Draw(HDC hdc) { assert(FALSE); };
};

class Line : public Shape
{
public:
        virtual void Draw(HDC hdc)
        {
                MoveToEx(hdc, begin.x, begin.y, (LPPOINT) NULL);
      LineTo(hdc, end.x, end.y);
        }
};

class Rect : public Shape
{
public:
        virtual void Draw(HDC hdc)
        {
                Rectangle(hdc, begin.x, begin.y, end.x, end.y);
        }
};
```

Полиморфия при съобщенията:

Съобщенията в Windows са няколко хиляди и са точно дефинирани.

Често използвани стандартни съобщения:

1. WM_INITDIALOG – един път при създаване на прозорец; начално инициализиране на данни; отваряне на файлове; заемане на оперативната памет; инстанциране на обекти.
2. WM_CLOSE – при затваряне на процеса;
3. WM_QUIT – затваря приложение;
4. WM_CHAR – съобщения от клавиатура (натиснат е клавиш);
5. WM_LBUTTONDOWN(UP) съобщения от мишката за натискане на ляв бутон (захват);
6. WM_RBUTTONDOWN(UP) съобщения от мишката за натискане на десен бутон (захват)
7. WM_MOUSEMOVE съобщения от мишката за преместване на обект (влачене, пречертаване);
8. WM_SIZE - съобщения при промяна размера на прозореца;
9. WM_PAINT – съобщение изпращано при изрисуване на главния прозорец, обработвано от UpdateWindow;
10. WM_DESTROY – съобщение за разрушаване на децата на главния прозорец;
11. WM_COMMAND – съобщение за въведена команда от меню или диалогов кутия;
12. WM_CREATE – съобщение за създаване на обект;

Пример: В един прозорец би могло да има следните съобщения: Подготовка на съдържанието на диалогова кутия с WM_INITDIALOG, извеждане на изображение на главния прозорец с WM_PAINT, команди въведени от менюто или диалогов прозорец с WM_COMMAND и затваряне на приложението с WM_DESTROY.

Функции за изпращане на съобщения

Send Message

NotifyMessage

PostMessage

```
LRESULT WINAPI SendMessage(HWND   hWnd,UINT
Msg, WPARAM wParam,LPARAM lParam );
```

Изпраща специфично съобщение на ОС или прозореца. Ивиква процедурата за

обработка на определен прозорец и не връща управлението на следващия оператор, без да изчаква съответната процедурата да е обработила съобщението. За да се изпрати съобщение и управлението да се върне без изчакване на обработката му се използват **SendMessageCallback** или **SendNotifyMessage** function. Поставянето на съобщение в опашката на потока и връщането на управлението веднага се използват функциите **PostMessage** или **PostThreadMessage**. .

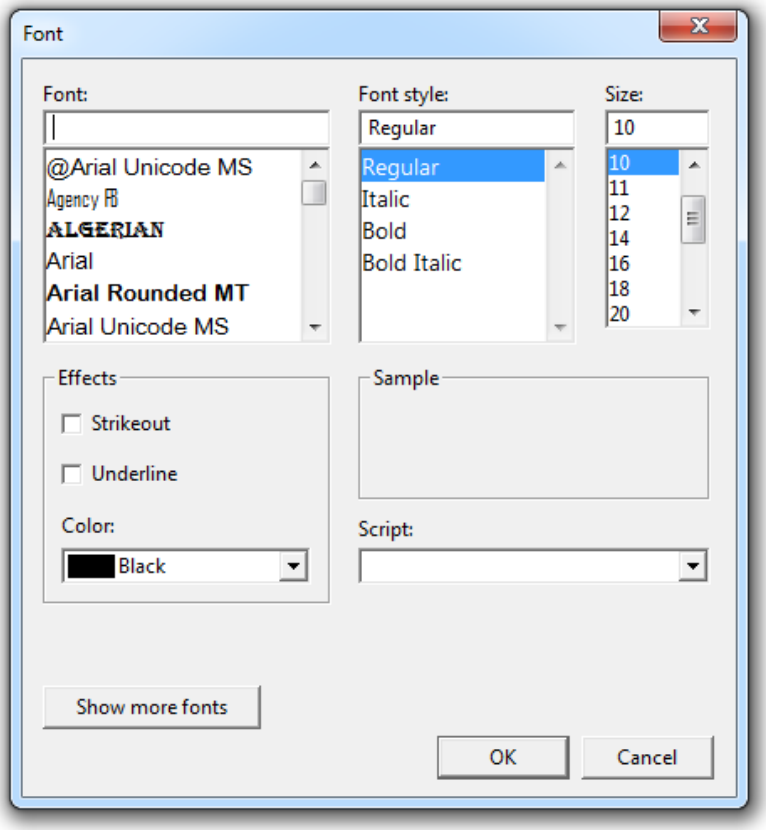
```
BOOL WINAPI SendNotifyMessage(HWND   hWnd,UINT
  Msg, WPARAM wParam,LPARAM lParam );
```

Изпраща специфични съобщения към ОС или прозоречната функция и предава управлението към следващия оператор без да изчаква обработката на съобщението.

```
BOOL WINAPI PostMessage(HWND   hWnd,UINT
  Msg, WPARAM wParam,LPARAM lParam );
```

Параметърът **Msg е от системния списък на съобщенията** (System-Defined Messages).

Създаване на приложение текстов редактор



```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
	int wmId, wmEvent;
	TCHAR szHello[MAX_LOADSTRING];
	LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

	static HWND hEdit;
	CHAR szText[] =   "Some text";

	UINT timer;
	POINT pt;
	static POINT ptOld;
	RECT rc;

	switch (message)
	{
		case WM_COMMAND:
			wmId    = LOWORD(wParam);
			wmEvent = HIWORD(wParam);
			// Parse the menu selections:
			switch (wmId)
			{
				case IDM_UNDO:
			  if(SendMessage(hEdit, EM_CANUNDO, 0, 0))
				SendMessage(hEdit, WM_UNDO, 0, 0);
			  break;
			case IDM_CUT:
			    SendMessage(hEdit, WM_CUT, 0, 0);
			    break;
			case IDM_COPY:
			    SendMessage(hEdit, WM_COPY, 0, 0);
			    break;
			case IDM_PASTE:
			    SendMessage(hEdit, WM_PASTE, 0, 0);
			    break;
			case IDM_DEL:
			    SendMessage(hEdit, WM_CLEAR, 0, 0);
			    break;
				case IDM_SEL:
					SendMessage(hEdit, EM_SETSEL, 0, -1);
					break;
				case IDM_ABOUT:
			  DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
(DLGPROC)About);
		break;
case IDM_FONT:
                                {
                                static HFONT hFont = 0;
                                LOGFONT lf;
                                CHOOSEFONT cf;
                                memset (&cf, 0, sizeof(cf));
                                cf.lStructSize = sizeof(cf);
                                cf.hwndOwner = hEdit;
                                cf.lpLogFont = &lf;
                                cf.Flags = CF_SCREENFONTS|CF_EFFECTS;

                                if (ChooseFont(&cf)){
                                        hFont = CreateFontIndirect(&lf);
                                SendMessage (hEdit,WM_SETFONT,(WPARAM) hFont,TRUE);
                                }}
```

```
                        break;
                    case IDM_EXIT:
                        DestroyWindow(hWnd);
                        break;
                    default:
                        return DefWindowProc(hWnd, message, wParam,
lParam);
                }
                break;
            case WM_CREATE:
hEdit = CreateWindow("EDIT", NULL, WS_CHILD | WS_VISIBLE | WS_VSCROLL |
ES_LEFT | ES_MULTILINE | ES_AUTOVSCROLL, 0, 0, 0, 0, hWnd, NULL, hInst,
NULL);
      SendMessage(hEdit, WM_SETTEXT, 0, (LPARAM) szText);
                GetCursorPos(&ptOld);
                return 0;
//    case WM_SETFOCUS:
//         SetFocus(hEdit);
//        return 0;
        case WM_SIZE:
            MoveWindow(hEdit, 0, 0, LOWORD(lParam), HIWORD(lParam), TRUE);
            return 0;
            case WM_DESTROY:
                PostQuitMessage(0);
                break;
            default:
                return DefWindowProc(hWnd, message, wParam, lParam); }
   return 0;
}
```

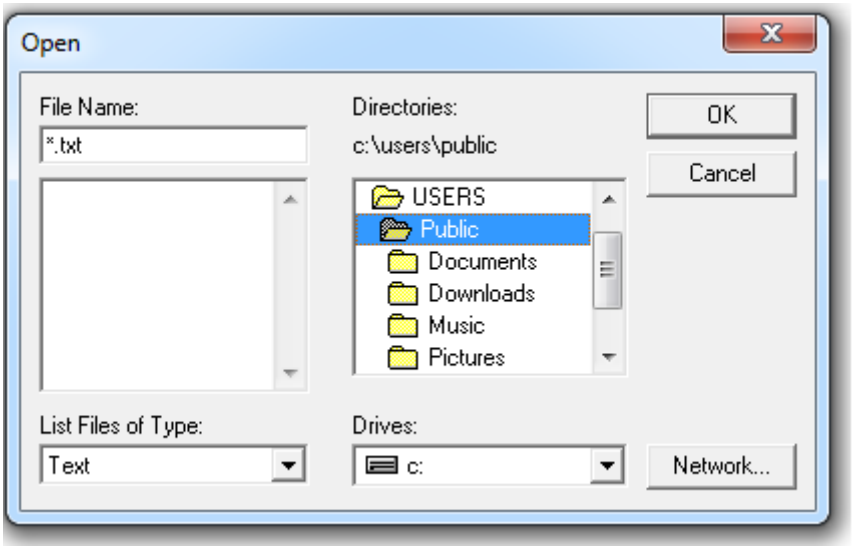
Работа с файлове

xxx.h

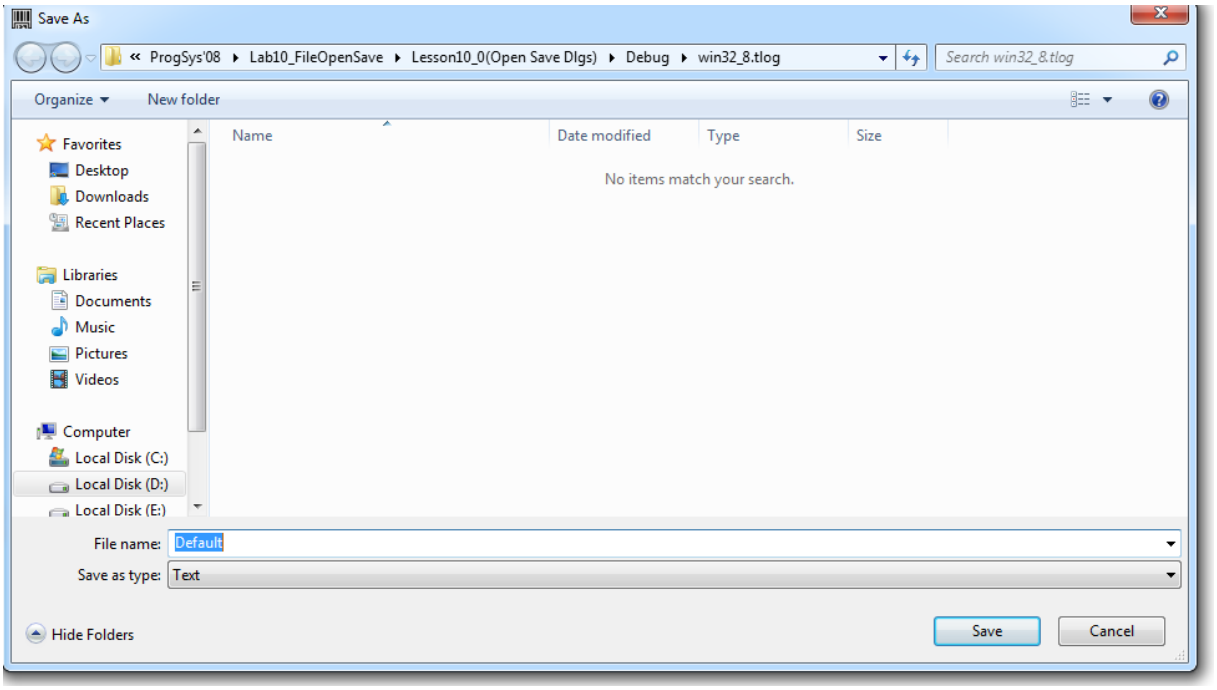
```
void OnOpenFile(HWND);
void OnSaveAsFile(HWND);
```

xxx.cpp

```
#include <stdio.h>
#include <Commdlg.h>

case IDM_OPEN:
      OnOpenFile(hWnd);
      break;
case IDM_SAVE_AS:
      OnSaveAsFile(hWnd);
      break;
```




```
void OnOpenFile(HWND hWnd)
{
	OPENFILENAME ofn;
	char szFilter[] = "All\0*.*\0Text\0*.txt\0";
	char szFile[MAX_PATH];
	char szFileTitle[MAX_PATH];

	sprintf(szFile, "");
	sprintf(szFileTitle, "");

	memset(&ofn, 0, sizeof(OPENFILENAME));
	ofn.lStructSize = sizeof(OPENFILENAME);
	ofn.hwndOwner = hWnd;
	ofn.lpstrFilter = (LPSTR)szFilter;
	ofn.nFilterIndex = 2;
	ofn.lpstrFile = (LPSTR)szFile;
	ofn.nMaxFile = sizeof(szFile);
	ofn.lpstrFileTitle = (LPSTR)szFileTitle;
	ofn.nMaxFileTitle = sizeof(szFileTitle);
	ofn.Flags = (OFN_FILEMUSTEXIST | OFN_HIDEREADONLY | OFN_PATHMUSTEXIST
|
		OFN_NOCHANGEDIR | OFN_ENABLETEMPLATE);
	ofn.lpTemplateName = MAKEINTRESOURCE(IDD_OPEN);

	if(GetOpenFileName(&ofn)==TRUE)
	{
		char szTemp[256];
		sprintf(szTemp, "Selected file is %s (full path: %s)",
			ofn.lpstrFileTitle, ofn.lpstrFile);
		MessageBox(hWnd, szTemp, "Info", MB_OK|MB_ICONINFORMATION);}}
```




```
void OnSaveAsFile(HWND hWnd)
{
	OPENFILENAME ofn;
	char szFilter[] = "All\0*.*\0Text\0*.txt\0";
	char szFile[MAX_PATH];
	char szFileTitle[MAX_PATH];

	sprintf(szFile, "Default");
	sprintf(szFileTitle, "");

	memset(&ofn, 0, sizeof(OPENFILENAME));
	ofn.lStructSize = sizeof(OPENFILENAME);
	ofn.hwndOwner = hWnd;
	ofn.lpstrFilter = (LPSTR)szFilter;
	ofn.nFilterIndex = 2;
	ofn.lpstrFile = (LPSTR)szFile;
	ofn.nMaxFile = sizeof(szFile);
	ofn.lpstrFileTitle = (LPSTR)szFileTitle;
	ofn.nMaxFileTitle = sizeof(szFileTitle);
	ofn.Flags = (OFN_FILEMUSTEXIST | OFN_HIDEREADONLY |
		OFN_PATHMUSTEXIST | OFN_NOCHANGEDIR);
	ofn.lpstrDefExt = "txt";
	if(GetSaveFileName(&ofn)==TRUE)
	{   char szTemp[256];
		sprintf(szTemp, "Selected file is %s (full path: %s)",
			ofn.lpstrFileTitle, ofn.lpstrFile);
		MessageBox(hWnd, szTemp, "Info", MB_OK|MB_ICONINFORMATION);
	}
}
```

Лекция 8

Графични функции в VC++

Update and Painting Methods

BeginPaint	Prepares the window for painting.
EndPaint	Marks the end of painting.
GetDC	Retrieves a device context for the client area.
GetDCEx	Retrieves a device context for the client area and allows clipping options.
GetUpdateRect	Retrieves the coordinates of the smallest rectangle that completely encloses the update region.
GetUpdateRgn	Retrieves the update region and copies it into a specified region.
GetWindowDC	Retrieves a device context for the entire window.
Invalidate	Invalidates the entire client area.
InvalidateRect	Invalidates the client area within the specified rectangle.
InvalidateRgn	Invalidates the client area within the specified region.
IsWindowVisible	Determines the window's visibility state.
LockWindowUpdate	Disables or enables drawing in the window.
Print	Requests that the window be drawn in a specified device context.
PrintClient	Requests that the window's client area be drawn in a specified device context.
RedrawWindow	Updates a specified rectangle or region in the client area.
ReleaseDC	Releases a device context.
SetRedraw	Sets or clears the redraw flag.
ShowOwnedPopups	Shows or hides the pop-up windows owned by the window.
ShowWindow	Sets the window's show state.
ShowWindowAsync	Sets the show state of a window created by a different thread.
UpdateWindow	Updates the client area.
ValidateRect	Validates the client area within the specified rectangle.
ValidateRgn	Validates the client area within the specified region.

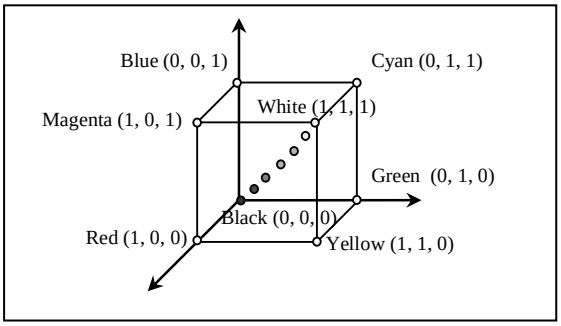
Основни средства

CreateSolidBrush	Initializes a brush with the specified solid color.
CreateHatchBrush	Initializes a brush with the specified hatched pattern and color.
CreateBrushIndirect	Initializes a brush with the style, color, and pattern specified in a LOGBRUSH structure. (**lbColor, lbHatch, lbStyle**)
CreatePatternBrush	Initializes a brush with a pattern specified by a bitmap.
CreateDIBPatternBrush	Initializes a brush with a pattern specified by a device-

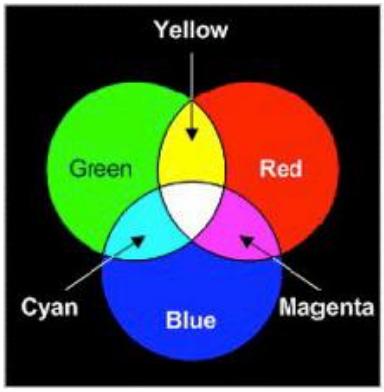
	independent bitmap (DIB).
CreateSysColorBrush	Creates a brush that is the default system color.

10 Цветови модел RGB (Red-Green-Blue)

Моделът е адитивен и е най-често използван в графичните формати. представлява тримерното цветовото пространството на единичен куб, върховете на който съответстват точно на основните и допълнителни цветове.



Фиг. 7.3. Цветови Модел RGB



Основните цветове са – червен (Red), зелен (Green)и син, използвани във всички дисплеи. Диагоналната линия между началото на координатната система и точката на бялото представлява сивия цвят във всички негови нюанси (Фиг.7.3.). Всеки пиксел се представя във вида на трикомпонентна величина. На непрекъснатия диапазон от стойности 0 до 1 от цветовия модел се съпоставя диапазон от дискретни стойности, всяка в интервала 0÷255. Те се задават съответно за интензивността на трите компоненти : червено, зелено и синьо. Примери на цветове дефинирани чрез модела **RGB**:

– с кодове зададени в ***десетична*** бройна система са: RGB(127,127,127) – сиво, чиито три компоненти имат еднакви стойности , червено - RGB(255,0,0), бяло - RGB(255,255,255), жълто - RGB(255,255, 0), кафяво - RGB(255,200,100);; розово - RGB(255,102, 228).

– с ***шестнадесетично*** зададени кодове на компонентите като примерите: жълто - RGB(0xFF,0xFF,0x00); розово - RGB(0xFF,0x66, 0xEF).

При извеждане на информация към екрана, принтера и др. тя не се извежда директно към хардуера, а се използва универсален интерфейс, наречено интерфейс на ***графично устройство или GDI (graphical device interface)*** . Windows предоставя драйверите за принтера и видео-драйверите. Вместо да адресира директно хардуера, програмата извиква GDI функции, които се обръщат към една структура от данни, наречена ***контекст на устройство*** (device context). Windows асоциира тази структура с физическото устройство и издава подходящите входно - изходни инструкции. Всеки път когато програмата чертае на дисплея или на принтера, тя използва GDI функциите. Това са функции които чертаят точки, линии, правоъгълници, многоъгълници, битмапи, текст. В GDI ключовият елемент е ***контекст на устройството*** (device context), който представлява дадено физическо устройство. Всеки C++

обект (от тип контекст на устройството) има асоцииран с него Windows – контекст на устройство. Той е асоцииран посредством 32-битов манипулатор (handle) от тип HDC.

1 1. Създаване на DC или зареждане на този асоцииран с прозореца

```
a) PAINTSTRUCT pc
HDC                 hdc
////////……
        case WM_PAINT:
        hDC=BeginPaint(hwnd, &pc);

            ///…… графични функции

        EndPaint(hwnd, &pc);
        break;


b) HDC                  hdc
//////……
        case WM_DRAW:
        hDC=GetDC(hwnd); //зареждане на контекста асоцииран с прозореца
             ///…… графични функции
        ReleaseDC(hwnd, hdc);
        break;
```

RECT **rt**; // type definition (RECT is standard type).
GetClientRect(hWnd, &**rt**); // getting of parameters of current rectangle in **rt.** rt.left, rt.rignh, rt.top, rt.boottom (integer values)

Относно hDC можем да получим информация за спецификата на устройството чрез функции като:

GetTextMetrix(hdc, &tm), където tm е размера на символите, примрно default font:

xChar=tm.tmAvecharWidth; yChar=tm.tmHeight;

или да изпълним графичните функции:

TextOut(hDC,ixPos,iYPos,szText,strlen(szText)); като например:

TextOut(hdc, 150, 100, bla, strlen(bla));

2 **Ползване н ацветове. Colors** (COLORREF ***crColor*** *зарежда цвета в три компонента)*
RGB(red components, green components, blue component) Компонентите са в интервала от 0 до 255.
Примери : red - RGB(255,0,0), white -RGB(255,255,255), yellow - RGB(255,102, 228), brown - RGB(255,200,100)

3 Извеждане на текст в графичен режим

DrawText(hdc, szHello, strlen(szHello), &rt, DT_CENTER) // рисува текста (szHello) на първия ред на **rt в средата чрез** DT_CENTER;
SetTextColor(hdc, RGB(190, 190, 190)); // задава цвят на текст в **hdc;**
SetBkColor(hdc, RGB(100, 100, 100)); // задава цвят на фона на текста;
TextOut(hdc, 600, 300, "text1", strlen("text1")); // рисува текста ("text1"), от точка (600;300) в handl **hdc.**

4 Рисуващи средства

Избор на средство:
SelectObject(**hdc**, **hPen**); or SelectObject(hdc, hB);// задава текущ обект за handle hdc.
DeleteObject(hPen); // освобождава и изтрива обекта.

- **Пера**

BOOL CreatePen(int *nPenStyle***, int** *nWidth***, COLORREF** *crColor* **);** The first version of **CreatePen** initializes a pen with the specified style, width, and color. The pen can be subsequently selected as the current pen for any device context.
HPEN hPen = **CreatePen**(PS_SOLID, 2, **RGB(0, 0, 0)**); // Create and **Set Pen**
The pen can subsequently be selected as the current brush for any device context.
SelectObject(**hdc**, **hPen**); // select object hPen in handle hdc.

- **Четки**

BOOL CreateSolidBrush(COLORREF *crColor* **);** Initializes a brush with a specified solid color. The brush can subsequently be selected as the current brush for any device context.

HBRUSH hBrush = **CreateSolidBrush**(RGB(132, 123, 231)); **// fiiling color**
SelectObject(hdc, hBrush);
BOOL CreateHatchBrush(int *nIndex***, COLORREF** *crColor* **);** Initializes a brush with the specified hatched pattern and color. The brush can subsequently be selected as the current brush for any device context.
Valie of nIndex maygh be folloung constants:

HS_BDIAGONAL Downward hatch (left to right) at 45 degrees
HS_CROSS Horizontal and vertical crosshatch
HS_DIAGCROSS Crosshatch at 45 degrees
HS_FDIAGONAL Upward hatch (left to right) at 45 degrees
HS_HORIZONTAL Horizontal hatch
HS_VERTICAL Vertical hatch

BOOL CreatePatternBrush(CBitmap* *pBitmap* **);** Initializes a brush with a pattern specified by a bitmap. The brush can subsequently be selected for any device context that supports raster operations. The bitmap identified by *pBitmap* is typically initialized.
The programer can use his own fill set with a resource **bitmap.** In following exzample this fill set by resourse with ID - IDB_BITMAP1.
HBITMAP hBmp=LoadBitmap(hInst, (const char*)IDB_BITMAP1); // loadoing of bitmap image in hBmp;
HBRUSH hBrush2=CreatePatternBrush(hBmp); /// user filling

5 Рисуване

Rectangle(hdc, Xleft, Ytop, Xrigh, Ybottom) // рисува правоъгълник по две срещуположни точки.
Пример - Rectangle(hdc, 120, 200, 250, 400);

Ellipse(hdc, Xcenter, Ycenter, RadiusX, RadiusY); // рисува елипса по две срещуположни точки.
Пример - Ellipse(hdc, 300, 150, 400, 500);
MoveToEx (hdc, X, Y, NULL); // Премества графичния курсор то точка (X,Y);.

LineTo(hdc, X,Y); // Рисува линия от точката на графичния курсор до т. (X,Y).
Пример - LineTo(hdc, rt.right-10, rt.bottom-10);

POINT p[n];
Polygon(hdc, p,n);// рисува полигон от n броя точки зададени в масив.

Пример: POINT pp[3]; Polygon(hdc, pp,4);

Поставяне на изображения от един DC в друг без промяна на размера му.

```
BOOL BitBlt(
HDC hdcDest,          // дескриптор na priemniq DC
  int nXOriginDest,     // x-коорд. горен ляв ъгъл на приемния правоъгълник
  int nYOriginDest,     // y-коорд. горен ляв ъгъл на приемния правоъгълник
  int nWidthDest,       // ширина на приемния правоъгълник
  int nHeightDest,      // височина на приемния правоъгълник.
  HDC hdcSrc,           // дескриптор входния DC
  int nXOriginSrc,      // x-коорд. горен ляв ъгъл на входния правоъгълник
  int nYOriginSrc,      // y-коорд. горен ляв ъгъл на входния правоъгълник

  DWORD dwRop           // код на растровата операция


);
Пример BitBlt(hDC,x,y,p,z,hdc,0,0,SRCCOPY);
```

Поставяне на изображения от един DC в друг с промяна на размера му.

```
BOOL StretchBlt(
  HDC hdcDest,          // дескриптор na priemniq DC
  int nXOriginDest,     // x-коорд. горен ляв ъгъл на приемния
правоъгълник
  int nYOriginDest,     // y-коорд. горен ляв ъгъл на приемния
правоъгълник
  int nWidthDest,       // ширина на приемния правоъгълник
  int nHeightDest,      // височина на приемния правоъгълник.
  HDC hdcSrc,           // дескриптор входния DC
  int nXOriginSrc,      // x-коорд. горен ляв ъгъл на входния
правоъгълник
  int nYOriginSrc,      // y-коорд. горен ляв ъгъл на входния
правоъгълник
  int nWidthSrc,        // ширина на входния правоъгълник
  int nHeightSrc,       // височина на входния правоъгълник
  DWORD dwRop           // код на растровата операция
);
```

Функция **StretchBlt** копира изображение от входен правоъгълник в друг, като го разтяга или свива съгласно режима в контекста на приемния DC.
Пример:

```
StretchBlt(hDC,x,y,p,z,hdc,0,0,85,74,SRCCOPY);
```

6 Пример за статична графика

…in WndProc(….. **case WM_PAINT:** { **hdc** = BeginPaint(hWnd, &ps);
// TODO: **Add any drawing code here**...
RECT **rt**; **GetClientRect**(hWnd, &**rt**); // get parameters of current rectangle in **rt.**
DrawText(hdc, szHello, strlen(szHello), &rt, DT_CENTER)
HPEN hPen = **CreatePen**(PS_SOLID, 2, **RGB(0, 0, 0)**); // **Set Pen**
SelectObject(hdc, hPen);
// **Painting 2 rectangles**
HBRUSH hBrush1=CreateSolidBrush(RGB(0xFF,0xFF,0x00));
SelectObject(hdc, hBrush1);
Rectangle(hdc, 134, 456, 324, 190);
Rectangle(hdc, 634, 456, 340, 190);
// **Fill ellipse with user witmap image**
HBITMAP hBmp=LoadBitmap(hInst, (const char*)IDB_BITMAP1);
HBRUSH hBrush2=CreatePatternBrush(hBmp);
SelectObject(hdc, hBrush2);
Ellipse(hdc, 134, 456, 324, 190);
//елипса
HBRUSH hBrush3=CreateHatchBrush(HS_CROSS,RGB(0x88,0x88,0x20));
SelectObject(hdc, hBrush3);
Ellipse(hdc, 634, 456, 340, 190);
// **ouputting text in graphical window**
char* bla=" I can paint ";
SetTextColor(hdc,RGB(0x5F,0x3F,0x00));
SetBkColor(hdc,RGB(0xFF,0x66, 0xEF)); // yellow RGB(255,255,0)
TextOut(hdc, 150, 100, bla, strlen(bla));
EndPaint(hWnd, &ps); break;}
//полигон

```
POINT pp[3];
HBITMAP hBmp1=LoadBitmap(hInst, (const char*)IDB_BITMAP4);
                    HBRUSH hBrush14=CreatePatternBrush(hBmp1);
                    SelectObject(hdc, hBrush14);
                    pp[0].x=760; pp[0].y=500;
                    pp[1].x=925; pp[1].y=380;
                    pp[2].x=990; pp[2].y=500;
                    pp[3].x=890; pp[3].y=600;
                    Polygon(hdc, pp,4);
```

7

8 Пример за рисуване с мишката при натиснат ляв бутон

```
case WM_LBUTTONDOWN: {  x = LOWORD(lParam);     y = HIWORD(lParam);     break;}
```

.....

```
case WM_MOUSEMOVE: if (wParam == MK_LBUTTON) {hdc = GetDC(hWnd);
                                    COLORREF color=RGB(0x10,0x10,0xF0);
                                    HPEN hPen=CreatePen(PS_SOLID, 5, color);
                                    SelectObject(hdc, hPen);
                                    MoveToEx (hdc, x, y, NULL); // мести GC до т. X,Y
                                    x=LOWORD(lParam);
                                    y=HIWORD(lParam);
                                    LineTo (hdc, x, y);}; break;
```

9 Пример за рисуване с цвят избран от палитра

```
case ID_FILE_NEW:
                                    {
```

```
                        // DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
(DLGPROC)About);
                        CHOOSECOLOR color;
                        COLORREF init =RGB(234,16,239);
                        static COLORREF
Arr[16]={RGB(234,16,239),RGB(234,16,239),RGB(234,16,239),RGB(234,16,239),RG
B(234,16,239),RGB(234,16,239),RGB(234,16,239),RGB(234,16,239),RGB(234,16,23
9),RGB(234,16,239),RGB(234,16,239),RGB(234,16,239),RGB(234,16,239),RGB(234,
16,239),RGB(234,16,239),RGB(234,16,239)};
                        memset (&color,0,sizeof(color));
                        color.lStructSize=sizeof(color);
                        color.hwndOwner=hDlg;
                        color.rgbResult=init;
                        color.lpCustColors=Arr;
                        color.Flags=CC_SOLIDCOLOR|CC_RGBINIT|CC_FULLOPEN;
                        if(ChooseColor(&color)){
                            color.rgbResult;
                  HDC hDC = GetDC(hDlg);
                            HBRUSH hBrush =
CreateSolidBrush(color.rgbResult);
                            SelectObject(hDC,hBrush);
                    Rectangle(hDC,0,0,100,100);

                      ReleaseDC(hDlg,hDC);

                        }
                            }

                        break;
```

11 Пример за задаване на размер на правоъгълник с мишката и поставянето на изображение в него

```
case WM_LBUTTONDOWN:
                        x=p=LOWORD(lParam);
                        y=z=HIWORD(lParam);
                        break;
            case WM_MOUSEMOVE:
                  if(wParam&MK_LBUTTON) {
      HDC hDC=GetDC(hDlg);
                        COLORREF color=RGB(1,1,1);
                        HPEN hPen=CreatePen(PS_DOT,1,color);
                        SelectObject(hDC,hPen);
                        SetROP2(hDC,R2_NOTXORPEN);
                        p=LOWORD(lParam);
                        z=HIWORD(lParam);
                        Rectangle(hDC,x,y,p,z);
                        ReleaseDC(hDlg,hDC);
                  }
                  break;
            case WM_LBUTTONUP:
      HDC hDC=GetDC(hDlg);
                        HDC hdc=CreateCompatibleDC(hDC);
                        HBITMAP hBit=LoadBitmap(hInst,(LPCTSTR)IDB_BOB);
                        SelectObject(hdc,hBit);
                        //BitBlt(hDC,x,y,p,z,hdc,0,0,SRCCOPY);
                        StretchBlt(hDC,x,y,p,z,hdc,0,0,85,74,SRCCOPY);
                  break;
```

12 Пример за движение на обект в прозорец

```
class Rect
{
public:
	Rect() {};
	~Rect() {};

	POINTS begin;
	POINTS end;

	void Draw(HDC hdc)
	{
		Rectangle(hdc, begin.x, begin.y, end.x, end.y);
	}

	void DrawAsSelected(HDC hdc)
	{
		HPEN newpen = CreatePen(PS_DOT, 0 , RGB(0, 0, 255));
		HPEN oldpen = (HPEN)SelectObject(hdc, newpen);
		Rectangle(hdc, begin.x, begin.y, end.x, end.y);
		SelectObject(hdc, oldpen);
		DeleteObject(newpen);
	}

	Rect* HitTest(POINTS point)
	{
		if(point.x >= begin.x && point.x < end.x &&
			point.y <= end.y && point.y > begin.y)
			return this;
		return NULL;
	}

	void MoveFromTo(POINTS from, POINTS to)
	{
		begin.x = begin.x + to.x - from.x;
		end.x = end.x + to.x - from.x;
		begin.y = begin.y + to.y - from.y;
		end.y = end.y + to.y - from.y;
	}
};

================================================================================
=================
static Rect rect1;
static Rect rect2;
static Rect* sel;
static POINTS anchor;
-----------------------------
	case WM_CREATE:
		rect1.begin.x = 100;
		rect1.begin.y = 50;
		rect1.end.x = 200;
		rect1.end.y = 150;

		rect2.begin.x = 300;
		rect2.begin.y = 300;
		rect2.end.x = 500;
		rect2.end.y = 450;
		break;
```

```
case WM_PAINT:
      hdc = BeginPaint(hWnd, &ps);
      rect1.Draw(hdc);
      rect2.Draw(hdc);
      EndPaint(hWnd, &ps);
      break;
case WM_LBUTTONDOWN:
      SetCapture(hWnd);
      sel = rect1.HitTest(MAKEPOINTS(lParam));
      if(sel == NULL)
            sel = rect2.HitTest(MAKEPOINTS(lParam));
      if(sel != NULL)
      {
            hdc = GetDC(hWnd);
            SetROP2(hdc, R2_NOTXORPEN);
            sel->Draw(hdc);
            sel->DrawAsSelected(hdc);
            ReleaseDC(hWnd, hdc);
            anchor = MAKEPOINTS(lParam);
      }
      return 0;
case WM_MOUSEMOVE:
      if((wParam & MK_LBUTTON) && (sel != NULL))
      {
            hdc = GetDC(hWnd);
            SetROP2(hdc, R2_NOTXORPEN);
            sel->DrawAsSelected(hdc);
            sel->MoveFromTo(anchor, MAKEPOINTS(lParam));
            sel->DrawAsSelected(hdc);
            anchor = MAKEPOINTS(lParam);
            ReleaseDC(hWnd, hdc);
      }
      break;
  case WM_LBUTTONUP:
      if(sel != NULL)
      {
            hdc = GetDC(hWnd);
            SetROP2(hdc, R2_NOTXORPEN);
            sel->DrawAsSelected(hdc);
            sel->Draw(hdc);
            sel = NULL;
            ReleaseDC(hWnd, hdc);
      }
      ReleaseCapture();
      return 0;
```

Лекция 9

Библиотека MFC (Microsoft Foundation Class Library)

Библиотека MFC е ***базов набор от класове***, написани на езика C++ и са предназначени за програмиране под Windows. Библиотеката представлява йерархия от класове на много нива с около ***200 члена*** и обезпечава създаването на приложения на основата обектно-ориентирания подход. Достойнствата на MFC е в това, че използването на нейните класове позволява да се избегне голяма част рутинна работа. Програмирането значително се улеснява, ***капсулирайки функциите*** на API ***с класовете*** от MFC. Интерфейсът, обезпечаван от библиотеката, практически не зависи от детайлите, които го реализират. Затова програмите, написани на основата на MFC, могат лесно да се адаптират за нови версии на Windows. MFC дава скелета, на чиято основа може да се създаде програма под Windows. Както всички приложения имат фиксирана структура, определена от **WinMain(),** така и приложенията създадени на основата на MFC са твърдо зададени.

Имената на класовете на библиотеката MFC започват с „C”, а имената на променливите от класа с „m_”. В най-простия случай на програма, написана с използването на MFC, се съдържат 2 класа : класа предназначен за създаване на ***приложението*** и класа за създаването на ***прозорец***. Първият (CApp) се поражда от класа **CWinApp** на библиотеката MFC, вторият обект (CMainWin) се поражда от **CFrameWnd.** Ядрото на MFC- приложенията глобалния обект **theApp** от класа **CApp**, отговаря за създаването на всички останали обекти и обработка на опашка от съобщения.

Създаването на MFC- програма включва следните действия:

- От **CFrameWnd** да се породи клас, определящ прозорец- клас **CMainWin**.
- От **CWinApp** да се породи клас, определящ приложението – **CApp**.
- Да се създаде карта на съобщенията.
- Да се пре-определи функцията **InitInstance()** на класа **CWinApp.**
- Да се създаде екземпляр от класа, определящ приложението – обекта **theApp**.

Примерен Листинг на минимална програма създадена на основата на класа библиотеки MFC.

```
#include <afxwin.h>
// Определение на основните класове
        // Клас за главния прозорец
class CMainWin: public CFrameWnd
{
public:
        CMainWin();
        DECLARE_MESSAGE_MAP()
};
        // Конструктор за прозорец
CMainWin:: CMainWin()
{
        Create(NULL, “Основно MFC приложение”
}
        // Клас за създаването на приложението
class CApp: public CWinApp
{
public: BOOL initInstance();
};
// Функция за инициализация на приложението
:BOOL  CApp:: initInstance()
{
        // Създаване на обекта – главен прозорец
        m_pMainWnd=new  CMainWin;
        // Изобразяване на прозорец
        m_pMainWnd->ShowWindow(m_nCmdShow);
        // Прерисуване на прозорец
        m_pMainWnd->UpdateWindow();
}
```

```
// Карта на съобщенията на приложенията
BEGIN_MESSAGE_MAP(CMainWin, CFrameWnd)
END_MESSAGE_MAP()
//////////////////////
// Начало на изпълнението на приложението
// Създаване на глобален обект – приложение
cApp  theApp;
//////////////////////
```

Текстът на програмата включва файла **AFXWIN.H*, който съдържа описание на класовете на библиотеката MFC и други библиотечни файлове.*** Приложението се изпълнява когато се създава екземпляр от класа CApp – обектът theApp. Породения клас CMainWin съдържа 2 члена: конструктор CMainWin() и макрос DECLARE_MESSAGE_MAP(), в който е обявена карта за обработка на съобщенията за класа CMainWin. Двойката команди BEGIN_MESSAAGE_MAP(CMainWin, CFrameWnd) и END_MESSAGE_MAP() заграждат картата на съобщенията на главния прозорец, като между тях трябва да се намира извикването на функцията за обработка на постъпващите съобщения. В примера не се обработват съобщения.

Обработка на съобщения

Съобщението е уникално 32-битово цяло значение. Във файла WINDOWS.H за всички съобщения са определени стандартни имена като WM_CHAR, WM_PAINT, WM_MOVE, WM_SIZE, WMLBUTTONDOWN и др. Често съобщенията се съпровождат с параметри носещи допълнителна информация (координати на курсора, код на натиснатия клавиш и др.) Всеки път когато произтича събитието, касаещо програмата ОС и изпраща съответното съобщение. Не е нужно да се описва реакцията на всяко съобщение. За съобщенията, които няма специална обработка, в MFC – програма се обработват по стандартен начин.

MFC съдържа пре-определящи функции – обработващи съобщенията, които могат да се използват в програмата. За организацията на обработката на съобщенията е необходимо да се изпълнят следните действия:

- Да се включат макрокомандите на съобщенията в картата на съобщенията на програмата;

- Да се включи в описанието на класа - прототип на функцията-обработчик на съобщението.
- Да се включи в програмата реализацията на функцията-обработчик.

Макрокомандите обезпечават извикване на функциите-обработчищи на стандартните съобщения на Windows. Имената на макрокомандите се състоят от префикса ON_, името на съобщението и двойка кръгли скоби. Например: **ON_WM_LBUTTONDOWN(), ON_WM_LBUTTONUP(), ON_WM_PAINT.** Предвижда се специална макрокоманда **ON_COMMAND()**, предназначенаа за съобщения, които постъпват при извикване на пунктове от меню, и съобщение определено от програмиста. Тази макрокоманда има два аргумента: **ON_COMMAND (ID, METHODNAME),** като първият аргумент – **идентификатор** на пункта от менюто или съобщението, определено от програмиста, а втория е **името на функцията-обработваща** съобщението.

Макрокомандата на съобщението се намира в картата на съобщения на прозореца. **Картата на съобщенията е специален механизъм, който свързва идентификатора на съобщения със съответния обработваща програма.** Едно приложение може да има няколко карти на съобщенията. Принадлежността на картата към някои прозорец се определя от параметъра на макроса **BEGIN_MESSAGE_MAP():**

```
BEGIN_MESSAGE_MAP(клас на прозореца-притежател, базов кклас)
// макрокоманди на съобщенията
END_MESSAGE_MAP()
```

Името на функцията – обработчик се състои от префикса **ON_и името** на съобщението (без WM_). Напр. **OnButtonDown().** Прототип на функцията – обработчик на съобщения трябва да е поместен в описанието на класа на прозореца пред макроса DECLARE_MESSAGE_MAP(). При описанието се използва спецификатор от типа **afx_msg**. Като пример да се прибави към по-горе показаната MFC – програма, обработка на съобщението **WM_LBUTTONDOWN**. След като потребителя натисне левия бутон в областта на прозореца, към приложението се изпраща съобщение WM_LBUTTONDOWN. При получаване на това съобщение програмата извежда на екран съобщение за случилото се събитие и координатите на точката, в

която е бил курсорът в този момент. За да се изпълни това се извършват следните действия:

- ***Модификация на картата*** на обработка на съобщенията

```
// карта на съобщенията
BEGIN_MESSAGE_MAP(CMainWin, CFrameWnd)
// макрокоманда на съобщението
    ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()
```

- ***Включване в описание на класа*** на прозореца на прототип на функцията – обработваща съобщението

```
class CMainWin: public CFrameWnd
{
public:
    CMainWin();
// прототип на функция – обработваща съобщението
// „натиснат ляв бутон”
afx_msg void OnLButtonDown (UINT, nFlags, CPoint point);
DECLARE_MESSAGE_MAP()
};
```

- В програмата се ***включва реализацията на функцията – обработваща съобщението***

```
void CMainWin::OnButtonLButtonDown(UINT nFlags, CPoint point)
{
    // CString – MFC class за работа с низове
    CString Str;
    Str.Format(“Press left button in point x=%d, y=%d, point.x, point.y);
    // MessageBox – метод на класа Cwnd – базов за CFrameWnd and CMainWin). Аргументи: (Str) текст на съобщението, заглавие на прозореца и флаг, определящ стила на прозореца.
    MessageBox(Str, “There is a message”, MB_OK|MB_ICONINFORMATION);
    // Стандартен обработваща програма
```

CFrameWnd::OnButtonDown(nFlags, point); }

Обработка на съобщения

Всяко обработвано съобщение трябва да се свързва с обработваща програма. .Тя е член функция на класа, приемащ съобщенията. Нейния прототип трябва предварително да се зададат. Като правило, името се състои от името на съобщението с префикс On. Например, за обработка на сообщението WM_CHAR - OnChar(), а за WM_LBUTTONDOWN - OnLButtonDown(). .

Пример, за обявяване на клас за обработка на съобщението WM_PAINT. Съобщението се изпраща към прозореца, когато той трябва да прерисува своята клиентска област.

```
Class CMainWin: public CFrameWnd {
public:
CMainWin();

afx_msg void OnPaint();

DECLARE_MESSAGE_MAP()

}
```

Спецификаторът afx_msg означава обявяване на обработчика на съобщения.

Всяка обработваща програма трябва бъде описана.

Пример на програма обработваща съобщения.

Програмата обработва натискането на левия и десния бутон на мишката, както и натискането на клавиш от клавиатурата. При натискане на левия бутон от мястото на курсора излиза текста „натисни дугия бутон”. За съобщенията WM_LBUTTONDOWN и WM_RBUTTONDOWN съответстват следните прототипи.

```
afx_msg void OnLButtonDown(UINT Flags, CPoint Loc);
afx_msg void OnRButtonDown(UINT Flags, CPoint Loc);
```

Параметри : **Flags** - натиснат ли е бутон на мишката, **Loc** - координати на курсора на мишката.. Клас CPoint се поражда от структурата POINT, определена така:

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

```
} POINT;
```

за съобщението WM_CHAR функцията има прототип:

```
afx_msg void OnChar(UINT Char, UINT Count, UINT Flags);
```

Първият параметър е ASCII-код символа, съответстващ на натиснатия клавиш. При натискане на несимволните клавиши съобщението WM_CHAR не се изпраща.

Контекст на устройствата

Device Context – структурата, обезпечаваща връзката на графичните функции с драйверите на конкретното устройство. В MFC има класове, капсулиращи DC.

Например, в MFC може да се получи контекст на клиентската област на прозореца създаващ екземпляр класа CClientDC. Конструкторът на този клас приема един параметър – указател за обекта-прозорец, **обикновено „this”.** След създаването на този обект може да се извежда графика в прозореца, използвайки функциите –членове на класа.

Сообщение WM_PAINT

Windows е устроена така, че за да се възстанови препокрита област, да се пречертае клиентската област се изпраща WM_PAINT. Обработката му води до обновяване на клиентската област. Прототип на обработ. пр. на WM_PAINT:

```
afx_msg void OnPaint();
```

Макрокомандата се нарича ON_WM_PAINT().
Пример : програмата извежда текста "Привет" в клиентската област от координ. x = 10, y = 20:

```
afx_msg void CMainWin::OnPaint()
{
CPaintDC paintDC(this);
paintDC.TextOut(10, 20, CString("Привет"));
}
```

В обработ. програма на WM_PAINT **винаги трябва да** се използва класа CPaintDC, който представлява класът на клиентската област, но предназначен за това съобщение.

Функция TextOut() извежда стринг в прозореца, от определена точка. Третият и параметър е CString. Този клас влиза в MFC.

Генерация на съобщението WM_PAINT

Принудителното изпращане на съобщението към прозореца в случай на пречертаване се осъществява с помощта на фукции със следващия прототип:

```
void CWnd::InvalidateRect(LPRECT pRegion,
                          BOOL EraseBackground = TRUE);
```

Параметърът pRegion задава област за пре-рисуване.. Параметърът EraseBackground определя, нужно ли е преди отправянето на съобщение с фон (по подразбиране бял).

Примерна програма

message handling.hpp

```
#include <afxwin.h>
// Клас на основния прозорец
class CMainWin: public CFrameWnd {
public:
CMainWin();
// Функции за обработки  на съобщенията
afx_msg void OnChar(UINT ch, UINT, UINT);
afx_msg void OnPaint();
afx_msg void OnLButtonDown(UINT flags, CPoint Loc);
afx_msg void OnRButtonDown(UINT flags, CPoint Loc);
// Допълнителни член-данни
char str[50];
int nMouseX, nMouseY, nOldMouseX, nOldMouseY;
char pszMouseStr[50];
// Декларация на картите за отклик на съобщенията
DECLARE_MESSAGE_MAP()
};
// Класът на приложението
class CApp: public CWinApp {
public:
BOOL InitInstance();
};
```

message handling.cpp

```
#include <afxwin.h>
#include <string.h>
#include "MESSAGE HANDLING.HPP"
// Реализация
BOOL CApp::InitInstance()
{
```

```
m_pMainWnd = new CMainWin;
m_pMainWnd->ShowWindow(SW_RESTORE);
m_pMainWnd->UpdateWindow();
return TRUE;
}
CMainWin::CMainWin()
{
// Създаване на основния прозорец
this->Create(0, "Обработка на съобщения");
// Инициализиране на променливите
strcpy(str, "");
strcpy(pszMouseStr, "");
nMouseX = nMouseY = nOldMouseX = nOldMouseY = 0;
}
// Реализация на картата на съобщенията на главния прозорец
BEGIN_MESSAGE_MAP(CMainWin /* класс */,
                  CFrameWnd /* базов клас */)
ON_WM_CHAR()
ON_WM_PAINT()
ON_WM_LBUTTONDOWN()
ON_WM_RBUTTONDOWN()
END_MESSAGE_MAP()

// Реализация на функциите обработващи съобщенията

afx_msg void CMainWin::OnChar(UINT ch, UINT, UINT)
{
sprintf(str, "%c", ch);
// Изпращане на  съобщение WM_PAINT и обновяване на прозореца
this->InvalidateRect(0);
}

afx_msg void CMainWin::OnPaint()
{
// Създаване на DC за  WM_PAINT
CPaintDC dc(this);
// Извеждаме текст
dc.TextOut(nOldMouseX, nOldMouseY,
" ", 30);
dc.TextOut(nMouseX, nMouseY, pszMouseStr);
dc.TextOut(1, 1, " ");
dc.TextOut(1, 1, str);
}

afx_msg void CMainWin::OnLButtonDown(UINT, CPoint loc)
{
nOldMouseX = nMouseX; nOldMouseY = nMouseY;
strcpy(pszMouseStr, "Натиснат ЛБ");
nMouseX = loc.x; nMouseY = loc.y;
this->InvalidateRect(0); // изпращаме съобщението WM_PAINT
```

```
}

afx_msg void CMainWin::OnRButtonDown(UINT, CPoint loc)
{  nOldMouseX = nMouseX; nOldMouseY = nMouseY;
strcpy(pszMouseStr, " Натиснат ДБ");
nMouseX = loc.x; nMouseY = loc.y;
this->InvalidateRect(0);     }

CApp App; // Единствен екземпляр на приложението
```

Съобщение WM_TIMER

За установяване на таймер се използва следната функция:

```
UINT CWnd::SetTimer(UINT Id, UINT Interval,
void (CALLBACK EXPORT *TFunc)(HWND, UINT, UINT, DWORD));
```

Ако се задададе като интервал - 0 мс, то таймерът преустановява своята работа.

Съобщенията от таймера се обработват с функцията:

```
afx_msg void OnTimer(UINT Id);
```

Съобщение WM_DESTROY

Обработва се с прототип: `afx_msg void OnDestroy();`

Отделянето на таймера става с функцията:

```
BOOL CWnd::KillTimer(int Id);
```

Функцията освобождава таймер с идентификатор Id.

Пример на програма за извеждане на текуща дата и време.

message handling II.hpp

```
#include <afxwin.h>
// Клас на основен прозорец
class CMainWin: public CFrameWnd {
public:
CMainWin();
afx_msg void OnPaint();
// Обраб. ф. за WM_DESTROY
afx_msg void OnDestroy();
// Обраб. ф. за  WM_TIMER
afx_msg void OnTimer(UINT ID);
char str[50];
```

```
DECLARE_MESSAGE_MAP()
};
// Клас на  приложението
class CApp: public CWinApp {
public:
BOOL InitInstance();
};
```

message handling II.cpp

```
#include <afxwin.h>
#include <time.h>
#include <string.h>
#include "MESSAGE HANDLING - II.HPP"
// Реализация
BOOL CApp::InitInstance()
{
m_pMainWnd = new CMainWin;
m_pMainWnd->ShowWindow(SW_RESTORE);
m_pMainWnd->UpdateWindow();

// Установяване на таймера с идентификатор 1 и интервал 500 мс

m_pMainWnd->SetTimer(1, 500, 0);
            return TRUE;    }

CMainWin::CMainWin()
{
// Определяне на правоъгълника на прозореца

RECT rect;
rect.left = rect.top = 10;
rect.right = 200;
rect.bottom = 60;

// Създаване на прозореца  в правоъгълника

this->Create(0, "CLOCK", WS_OVERLAPPEDWINDOW, rect);
strcpy(str, "");
}

// Реализация  на картата на Съобщенията на главния прозорец
BEGIN_MESSAGE_MAP(CMainWin, CFrameWnd)
ON_WM_PAINT()
ON_WM_DESTROY()
ON_WM_TIMER()
END_MESSAGE_MAP()
afx_msg void CMainWin::OnPaint()
{
CPaintDC dc(this);
```

```
// Въвеждане на текст за текущото време
dc.TextOut(1, 1, " ", 3);
dc.TextOut(1, 1, str);
}

afx_msg void CMainWin::OnTimer(UINT ID)
{

// Получаване на реда с текущото време
CTime curtime = CTime::GetCurrentTime();
tm *newtime;
newtime = curtime.GetLocalTm();
sprintf(str, asctime(newtime));
str[strlen(str) - 1] = '\0';

// Изпращане на съобщение  WM_PAINT
this->InvalidateRect(0);
}

afx_msg void CMainWin::OnDestroy()
{ //При затваряне на прозореца се спира и таймера.
        KillTimer(1);
}
CApp App; // Единствен екземпляр на приложението
```

Включване на меню в приложението

За да се включи меню в прозрец, то трябва ресурса да се укаже при създаването на прозореца, като последен параметър във функция Create():

```
this->Create(0, "Приложение с меню",
        WS_OVERLAPPEDWINDOW,
          rectDefautl, 0, "MYMENU");
```

За обработката му трябва да се създаде обраб. прог. за съобщението WM_COMMAND и всеки пункт от менюто.

Съобщение WM_COMMAND

За обработка се използва следната макрокоманда:

```
ON_COMMAND(Идентификатор, Име на Обраб. програма );
```

Всяка програма за WM_COMMAND е длъжен да връща void. Те нямат параметри

Акселератори

Таблицата на акселераторите може да се зареди с помощта на функция с прототип:

```
BOOL CFrameWnd::LoadAccelTable(LPCSTR ResourceName);
```

Прозорец със съобщения

За тези прости прозорци се използва функция с прототип.

```
int CWnd::MessageBox(LPCSTR MessageText,
LPCSTR WindowTitle = 0,
UINT MessageBoxType = MB_OK);
```

Пример программы, использваща меню, акселератори и прозорец със съобщения

menu.hpp

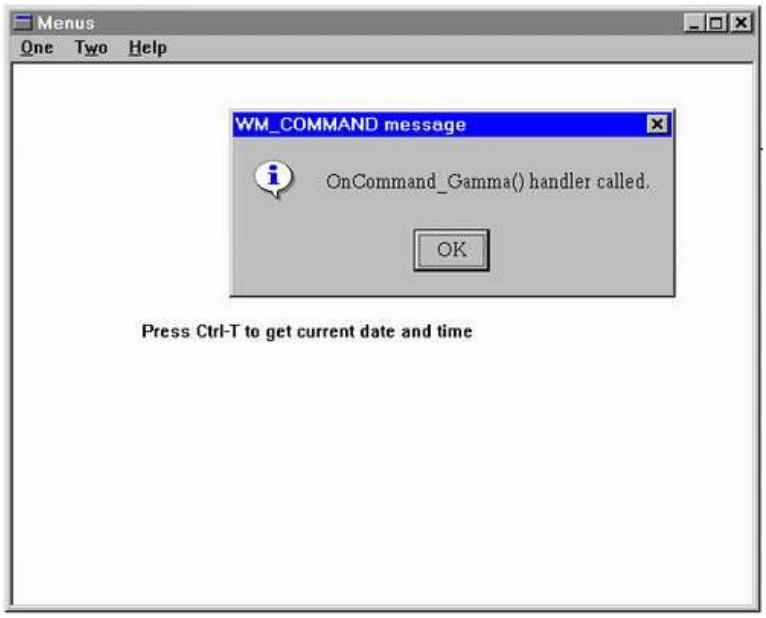
```
#include "stdafx.h"
// Клас Осн. прозорец
class CMainWin: public CFrameWnd {
public:
CMainWin();
afx_msg void OnPaint();
// прототипи
afx_msg void OnCommand_Alpha();
afx_msg void OnCommand_Beta();
afx_msg void OnCommand_Gamma();
afx_msg void OnCommand_Epsilon();
afx_msg void OnCommand_Zeta();
afx_msg void OnCommand_Eta();
afx_msg void OnCommand_Theta();
afx_msg void OnCommand_Help();
afx_msg void OnCommand_Time(); //само за акселератор
DECLARE_MESSAGE_MAP()
};

// Клас на приложението
class CApp: public CWinApp {
public:
BOOL InitInstance();
};
```

menu.cpp

```
#include "stdafx.h"
#include <string.h>
#include <stdio.h>
#include "MENU.HPP"
#include "IDS.H"

CMainWin::CMainWin()
{
// Правоъгълник на прозореца
RECT rect;
rect.left = 20; rect.top = 10;
rect.right = 600; rect.bottom = 460;
this->Create(0, "Menus", WS_OVERLAPPEDWINDOW, rect, 0,
                        "MYMENU");
// Зареждане на акселераторите
this->LoadAccelTable("MYMENU");
}

BOOL CApp::InitInstance()
{
m_pMainWnd = new CMainWin;
m_pMainWnd->ShowWindow(SW_RESTORE);
m_pMainWnd->UpdateWindow();
return TRUE;
}

// Карта обработка на сообщения
BEGIN_MESSAGE_MAP(CMainWin, CFrameWnd)
ON_WM_PAINT()
ON_COMMAND(IDM_ALPHA, OnCommand_Alpha)
ON_COMMAND(IDM_BETA, OnCommand_Beta)
ON_COMMAND(IDM_GAMMA, OnCommand_Gamma)
ON_COMMAND(IDM_EPSILON, OnCommand_Epsilon)
ON_COMMAND(IDM_ZETA, OnCommand_Zeta)
ON_COMMAND(IDM_ETA, OnCommand_Eta)
ON_COMMAND(IDM_THETA, OnCommand_Theta)
ON_COMMAND(IDM_HELP, OnCommand_Help)
ON_COMMAND(IDM_TIME, OnCommand_Time)
END_MESSAGE_MAP()

afx_msg void CMainWin::OnPaint()
{
CPaintDC dc(this);
CString s("Press Ctrl-T to get current date and time");
dc.TextOut(100, 200, s);
}

// Обработките на WM_COMMAND.
```

```
afx_msg void CMainWin::OnCommand_Alpha()
{
this->MessageBox("OnCommand_Alpha() handler called.",
                        "WM_COMMAND message",
                        MB_OK | MB_ICONINFORMATION);
}

afx_msg void CMainWin::OnCommand_Beta()
{
this->MessageBox("OnCommand_Beta() handler called.",
                        "WM_COMMAND message",
                        MB_OK | MB_ICONINFORMATION);
}

afx_msg void CMainWin::OnCommand_Gamma()
{
this->MessageBox("OnCommand_Gamma() handler called.",
                        "WM_COMMAND message",
              MB_OK | MB_ICONINFORMATION);
}

afx_msg void CMainWin::OnCommand_Epsilon()
{
this->MessageBox("OnCommand_Epsilon() handler called.",
"WM_COMMAND
message",
MB_OK | MB_ICONINFORMATION);
}

afx_msg void CMainWin::OnCommand_Zeta()
{
this->MessageBox("OnCommand_Zeta() handler called.",
                        "WM_COMMAND message",
              MB_OK | MB_ICONINFORMATION);
}


afx_msg void CMainWin::OnCommand_Eta()
{
this->MessageBox("OnCommand_Eta() handler called.",
                        "WM_COMMAND message",
              MB_OK | MB_ICONINFORMATION);
}

afx_msg void CMainWin::OnCommand_Theta()
{
this->MessageBox("OnCommand_Theta() handler called.",
              "WM_COMMAND message",
              MB_OK | MB_ICONINFORMATION);
}
```

```
afx_msg void CMainWin::OnCommand_Help()
{
this->MessageBox("OnCommand_Help() handler called.",
            "WM_COMMAND message",
            MB_OK | MB_ICONINFORMATION);
}

afx_msg void CMainWin::OnCommand_Time()
{
// Текущо време използвайки клас MFC CTime
CTime currentTime = CTime::GetCurrentTime();
CString s = currentTime.Format("%A %B %#d, %Y, %#I:%M%p");

s = "OnCommand_Time() handler called.\n\n"
   + ("Current date and time:\n" + s);
this->MessageBox(s, "WM_COMMAND message");
}
CApp App; // Единствен обект на приложението.
```



Диалози - Класове MFC за контроли

В MFC се съдържат класове за всички стандартни елементи за управления . Те са породени от класа CWnd. Основни класове за управления:

Класс	Елемент на управление
CButton	Бутони, превключватели
CEdit	Поле за възвеждане
CListBox	Списъци

CComboBox	Комбинирани спосъци
CScrollBar	Плъзгачи
Cstatic	Статически елементи

Класс CDialog

В MFC всички диалози са екземпляри на класа Cdialog или породени от него. Само най-простите диалози използват непосредствено класс CDialog. В общия случай е необходимо да се определи собствен клас. Клас CDialog има конструктори със следващите прототипи:

```
CDialog::CDialog(LPCSTR ResourceName, CWnd *Owner = 0);
CDialog::CDialog(UINT ResourceID, CWnd *Owner = 0);
CDialog::CDialog();
```

Параметърът ResourceName или ResourceID определят идентификатора на ресурса на диалога. Параметъра Owner – е указател на прозореца -собственик, ако е равен на 0, то собственикът е главния прозорец.

Обработка на съобщения от диалози

Всички диалози са разновидност на прозорец. За всеки прозорец се образува опашка от съобщения и те се обработват като тези към главния прозорец.

При всяко активиране на елемент на управление на диалога се изпраща съобщение WM_COMMAND. За обработка на това съобщение в картата на съобщенията на диалога трябва да се сложат макроси ON_COMMAND().

Извикване на диалога

Извиква се с член-функцията DoModal().

Резултатът ще бъде изобразяването на модален диалог. Прототип на функциията

```
virtual int CDialog::DoModal();
```

Функцията връща код на завършване.

Затваряне на диалога

По подразбиране диалогът се затваря при получаване на съобщения с идентификатори IDOK или IDCANCEL. Класът CDialog съдържа вградени обработки за два случая OnOK() и OnCancel(). За програмно затваряне на диалога е нужно да се извика член-функцията с прототип:

```
void CDialog::EndDialog(int RetCode);
```

Примерна програма с диалогов прозорец

Пример с меню, диалог, и преопределяне на функция OnCancel(), за да има и прозорец с потвърждаване. Диалогът се затваря програмно с возвръщане на код на завършване 7, който се връща от функция DoModal() и се изобразява в прозореца.

resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Resources.rc

#define IDC_RED 1007
#define IDC_GREEN 1008
#define IDM_DIALOG 40001
#define IDM_HELP 40002
#define IDM_EXIT 40003

// Next default values for new objects
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NO_MFC 1
#define _APS_3D_CONTROLS 1
#define _APS_NEXT_RESOURCE_VALUE 106
#define _APS_NEXT_COMMAND_VALUE 40010
#define _APS_NEXT_CONTROL_VALUE 1008
#define _APS_NEXT_SYMED_VALUE 101
#endif
#endif
```

Dialogs.hpp

```
#include "stdafx.h"
#include "resource.h"
// Клас на главния прозорец
class CMainWin: public CFrameWnd {
public:


// Конструктор
        // Title --, HowShow -- код на показване на прозореца
        CMainWin(CString Title, int HowShow);
        afx_msg void OnCommand_Dialog();
        afx_msg void OnCommand_Exit();
        afx_msg void OnCommand_Help();
        afx_msg void OnPaint();
        // Установяване на текст1
        virtual void SetInfoStr(CString str);
        private:
        CString infoStr;
```

```
        DECLARE_MESSAGE_MAP()
};
// Клас на приложението
class CApp: public CWinApp {
public:
BOOL InitInstance();
};
// Клас на диалоговия прозорец
class CSampleDialog: public CDialog {
public:
// DialogName -- идентификатор на ресурса на диалога
// Owner – прозорец -собственик (ако е NULL, то главния пр.)
CSampleDialog(char *DialogName, CWnd *Owner = NULL);
afx_msg void OnCommand_Red();
afx_msg void OnCommand_Green();
afx_msg void OnCancel();
private:
DECLARE_MESSAGE_MAP()
};

Dialogs.cpp
#include "stdafx.h"
#include "Dialogs.hpp"
#include "resource.h"


CApp App; // Единствен екземпляр на приложението
//-------------------------------------------------------

CMainWin::CMainWin(CString Title, int HowShow)
:infoStr("")
{
RECT rect;
rect.top = 10; rect.left = 50;
rect.bottom = 460, rect.right = 630;
this->Create(0, Title, WS_OVERLAPPEDWINDOW, rect, 0,
"DIALOGMENU");
this->ShowWindow(HowShow);
this->UpdateWindow();
this->LoadAccelTable("DIALOGMENU");
}

afx_msg void CMainWin::OnCommand_Dialog()
{
CSampleDialog sampleDialog("SAMPLEDIALOG", this);
int result = sampleDialog.DoModal();
char s[20];
sprintf(s, "%i", result);
this->SetInfoStr(infoStr + ". " +
"Функция CDialog::DoModal() върна " + CString(s));
```

```
}
afx_msg void CMainWin::OnCommand_Exit()
{
this->MessageBox("Завършване на приложението.",
                 "Dialogs", MB_ICONEXCLAMATION);
this->SendMessage(WM_CLOSE);
}
afx_msg void CMainWin::OnCommand_Help()
{
this->MessageBox("Программа, использующая диалог.\n"
                 "Написана с помощью MFC 4.0.",
                 "Dialogs", MB_ICONINFORMATION);
}

afx_msg void CMainWin::OnPaint()
{
CPaintDC paintDC(this);
paintDC.TextOut(10, 10, infoStr);
}

void CMainWin::SetInfoStr(CString str)
{
infoStr = str;
this->InvalidateRect(0);
}

// Карта на отклиците на сообщенията на  главния прозорец
BEGIN_MESSAGE_MAP(CMainWin, CFrameWnd)
ON_COMMAND(IDM_DIALOG, OnCommand_Dialog)
ON_COMMAND(IDM_HELP, OnCommand_Help)
ON_COMMAND(IDM_EXIT, OnCommand_Exit)
ON_WM_PAINT()
END_MESSAGE_MAP()

//------------------------------------------------
BOOL CApp::InitInstance()
{
m_pMainWnd = new CMainWin("Dialogs - приложение с диалог",
                          SW_RESTORE);

return TRUE;
}
```

```
//-------------------------------------------------
CSampleDialog::CSampleDialog(char *DialogName, CWnd *Owner)
:CDialog(DialogName, Owner)
{
}

afx_msg void CSampleDialog::OnCommand_Red()
{
// Член-функция CWnd::GetOwner() връща  указател на прозореца собственик
((CMainWin *) (this->GetOwner()))
->SetInfoStr("Натиснато е бутон 'Red');
}
afx_msg void CSampleDialog::OnCommand_Green()
{
        ((CMainWin *) (this->GetOwner()))
->SetInfoStr("Натиснат е бутон ‘'Green');
}

afx_msg void CSampleDialog::OnCancel()
{
((CMainWin *) (this->GetOwner()))
->SetInfoStr("Нажата кнопка 'Cancel'");

// Затваряне на диалог,ако е потвърдено намерението, значение 7 ще е върнато от DoModal().
int response = this->MessageBox("Потвърди?",
                                "Cancel", MB_YESNO);
if(response == IDYES)
this->EndDialog(7);
}

// Карта на отклиците на съобщенията към диалога
BEGIN_MESSAGE_MAP(CSampleDialog, CDialog)
ON_COMMAND(IDC_RED, OnCommand_Red)
ON_COMMAND(IDC_GREEN, OnCommand_Green)
END_MESSAGE_MAP()
//-------------------------------------------------
```

Базови класовете на библиотеката MFC

CView, CEditView, CFormView, CListView, CHtmlView, CRichView, CScrollView, CTreeView.

Инструментални средства.

Средата за разработка на програми Microsoft Visual C++ **Интегрирана развойна среда MS Visual C++** (Integrated Development Environment – (IDE)) е предназначена за създаването на приложения – програми, снабдени с всичко

необходимо за тяхната работа. В Microsoft Visual C++ приложенията често се разработват на основата на Microsoft Foundation Class Library (MFC) – библиотека от базови класове обекти на фирмата Microsoft, макар че има възможност и за създаването и на други типове програми. Такива приложения представляват съвкупност от обекти, които самото приложение и неговите компоненти: документи, вид на документите, диалози и т.н.

Всяко приложение взаимодейства с Windows чрез Application Programming Interface (API). API съдържа няколко стотици функции, които реализират всички необходими системни действия, такива като отделяне на памет, създаване на прозорци, извеждане на екран и т.н. Функциите API се съдържат в динамични библиотеки Dynamic Link Libraries (DLL), които се зареждат в паметта само в този момент, когато към тях има обръщение. Едно от подмножествата на API е Graphic Device Interface (GDI) – интерфейса на графични устройства. Задачата на GDI е да обезпечи апаратната независимост на графиката. На GDI се дължи това, че приложенията могат да се изпълняват на различни компютри.

Интерфейсът на Microsoft Visual C++ 6.0 включва удобни средства за създаването и редактирането на всички типове ресурси. Такива са мощното средство за скелетни приложения (AppWizard), а така също средството за създаване и управление на класовете на приложението (ClassWizard). Неоценима помощ при разработката на програми може да укаже библиотеката знания Microsoft Developer Network (MSDN). В тази библиотека могат да се намери информация не само за API и MFC , но и много примери.

Нови концепции в Windows. Нов изпълняем формат: xxx.EXE

NEW EXECUTABLE FORMAT

Отнася за :

- приложения : *.EXE;
- динамични библиотеки *.DLL;
- някои двоични файлове: шрифтове;
- драйвери за отделните устройства.

Допълнителна информация в новия формат:

- секция (Entry Table) на входните точки: име, номер, вх. адрес на всички експортируеми функции: (ако е EXE има само главната програма WinMain, която да се извиква от Windows);
- секция на всички импортируеми функции: функции, дефинирани в други модули;
- експортни и импортни функции се описват във файла: *.DEF (име и номер).

Лекция 10

Работа с динамична библиотека (DLL)

Андрей Уваров, http://andy.uvarov.ru/

От създаването си операционата система Windows използва библиотеки с динамично зареждане DLL (Dynamic Link Library), в които се съдържат реализации на най-често използваните функции.

Аспекти на създаването и използването на библиотеките DLL:

- как статично да се включат библиотеките DLL;
- как динамично да се зареждат библиотеките DLL;
- как да се създадат библиотеките DLL;
- как да се създават расширения на MFC с библиотеки DLL.

Използване на DLL

Практически е невъзможно да се създаде приложение на Windows, в което да не се използват библиотеки DLL. В DLL се съдържат всички функции на Win32 API и много други функции на операционната система Win32.

Общо говорейки DLL – са набор от функции, събрани в библиотеки. За разлика от статичните библиотели (файлове . lib), библиотеките DLL не са присъединени непосредствено към изпълнимия файл с помощта на свързващия редактор.

В изпънимия файл има информация само за метонахождението на библиотеката. В момента на изпълнението се зарежда цялата библиотека. Благодарение на това различни процеси могат да ползват съвместно едни и същи библиотеки, намиращи се в паметта. Такъв подход позволява да се съкрати обема на паметта, необходим за няколко приложения, използващи много общи библиотеки, а така също да се контролират размерите на EXE-файловете.

Ако библиотеката се използва само от едно приложение, то по-добре да се направи статична. Ако функциите от библиотеката влизат в състава само на една програма , то те могат да се вместят в текста на приложението.

Най-често проектите прикачват DLL статически, или неявно, на етапа на свързване. Зареждането на DLL при изпълнението на програмата се управлява от операционната система. DLL може да се зареди и явно, или динамически, в хода на работата на приложението.

Библиотеки за импортиране

При статично прикачване на DLL, то име.lib-файла се задава с параметрите на свързващия редактор в команден ред или в "Link" диалоговия прозрец на "Project Settings" в средата на Developer Studio. Файл име.lib, използван **при неявно включване на DLL,** - то това не е обикновенна статична библиотека. Такива .lib-файлове се наричат **библиотеки за импортиране** (import libraries). В тях се съдържат не самия код на библиотеките, а само препратките към всички функции, експортирани от DLL файла, в които и те се съхраняват. Като резултат импортиращите библиотеки имат по-малък размер, отколкото DLL файловете. Способите на тяхното създаване ще се разгледат по-долу.

Съгласуване на интерфейса

При използоване на собствени библиотеки или такива на независими разработчици, то се налага да се съгласува извикването на функциите с техния прототип.

По подразбиране в Visual C++ интерфейсите на функциите са съгласувани по правилата на C++. Това означава, че параметрите се прехвърлят в стек от дясно на ляво, като извикващата програма отговаря за тяхното отделяне от стека при излизане от функцията и разширенията на нейното име. Разширението на имената (name mangling) позволява на св. редактор да различава функции с еднакви имена, но различни списъци с аргументи.

Ако е необходимо да се вкючи библиотека на C към приложение на C++, всички функции от тази библиотека трябва да се обявявт като външни.

```
extern    "C" int MyOldCFunction(int myParam);
```

Модификаторът extern "C" може да се приложи и към цял блок, към който с помощта на директивата #tinclude е включен файл със старо заглавие C. Така, вместо модификации на всяка функция по отделно може да се използва:

```
extern    "C"
{
   #include "MyCLib.h"
}
```

Неявно включване на DLL

При пускане даденото приложение се опитва да намери всички файлове DLL, които са неявно включени към приложението и да ги постави в област на ОП заета от дадения процес. **Търсенето на DLL файлове ОС осъществява в следната последователност.**

- **в каталога, в който се намира EXE-файла.**
- **в текущия каталог на процеса.**
- **в системния каталог на Windows.**

Ако библиотеката DLL не е намерена, приложението извежда диалогов прозорец със съобщение за това. Ако библиотеката е намерена, то тя се помества в ОП на процеса, където и остава до неговото свършване. Така приложението може да се обръща към функциите в DLL.

Динамично зареждане и отделяне на DLL

Вместо това, Windows да изпълнява динамично свързване с DLL, може да се свърже програмата с модулите на библиотеката при изпълнението на програмата, като така в процеса на създаване на приложението не е нужно използването на библиотека за импортиране. В частност, може да се определи коя от библиотеките DLL е достъпна за ползвателя, или да се разреши на позвателя да избере, коя библиотека да се зареди. Така може да се използват различни DLL, в които са реализирани едни и същи функции, изпълняващи различни действия.

Зареждане на обикновена DLL

Първо е необходимо да се помести модула на библиотеката в паметта на процеса. Тази операция се изпълнява с помщта на функцията **::LoadLibrary**, имаща единствен аргумент – името на зареждания модул. Програмния фрагмент ще изглежда така:

```
HINSTANCE hMyDll;
::
if((hMyDll=::LoadLibrary("MyDLL"))==NULL) { /* не се зареди DLL */ }
else { /* приложението има право да използва функциите на DLL чрез hMyDll */ }
```

Стандартното разширение на файла с библиотеката, Windows счита .dll, ако не е указано друго. Ако в името е указана следата, само той ще се ползва за търсене. В противен случай Windows ще търси файл по същата схема, както и в случай на неявно включване, запозвайки от каталога от който се зарежда EXE-файла и продължавайки в съответствие със зачениятa на PATH.

Когато Windows намери файла, неговата пълна следа ще бъде сравнена с пътя на заредените вече библиотеки от дадения процес. Ако се намери тъждество, вместо зареждане на копието то към приложениео се връща дескриптора на заредената вече библиотека.

Ако файла е намерен и библиотеката е заредена, функцията **::LoadLibrary** връща нейния дескриптор, който се използва за достъп до функциите на библиотеката.

Преди да се използва библиотеката трябва да се получи нейния адрес. За това се използва директивата **typedef** за определяне типа на указателя на функцията и да се определи променлива от този нов тип, например:

```
// тип PFN_MyFunction ще бъде обявен за указател към функцията,
приемащ указател на символния буфер и значение от тип int
typedef int (WINAPI *PFN_MyFunction)(char *);
::
PFN_MyFunction pfnMyFunction;
```

После трябва да се получи дескриптор на библиотеката с помощта на който и се определя адреса на функцията, например адрес на функция с име MyFunction:

```
hMyDll=::LoadLibrary("MyDLL");
pfnMyFunction=(PFN_MyFunction)::GetProcAddress(hMyDll,"MyFunction");
::
int iCode=(*pfnMyFunction)("Hello");
```

Адресът на функцията се определя с помощта на функцията **::GetProcAddress**, като на нея се предава името на библиотеката и името на функцията.

Възможно е да се обръщаме към функция по нейния пореден номер, по който тя да се експортира. (при това за създаване на библиотека трябва да се използва def-файл):

```
pfnMyFunction=(PFN_MyFunction)::GetProcAddress(hMyDll,
        MAKEINTRESOURCE(1));
```

Премахване на библиотеката от паметта на процеса става с функцията ::**FreeLibrary**:

```
::FreeLibrary(hMyDll);
```

С помощта на функцията **LoadLibrary** може да се зареждат в паметта и изпълними файлове, без да се пускат в изпълнение. Дескрипторът на изпълнявания модул може след това при обръщане с функциите **FindResource** и **LoadResource** да се използва за търсене и зареждане на ресурсите на приложението. Отделянето на изпълним модул от паметта става с функцията **FreeLibrary**.

Пример за DLL и способите за зареждане

Код за динамно включвана библиотека, която се нарича MyDLL и съдържа една функция MyFunction, която само извежда едноо съобщение.
В хедърния файл се определя макроконтстанта EXPORT. По този начин се съобщава на свързващия редактор, че тази функция ще е достъпна за

използване от други програми, в резултат на което той прави запис за нея в библиотеката за импорта. Освен това такава функция, точно така както и прозоречната процедура трябва да се определя с помощта на константы **CALLBACK**:

MyDLL.h

```
#define EXPORT extern "C" __declspec (dllexport)
EXPORT int CALLBACK MyFunction(char *str);
```

Файлът на библиотеката също така се различава от обикновените файлове на C++ за Windows. Вместо функциията **WinMain** то има функция **DllMain**. Тази функция се използва за изпълнение на инициализация. За това библиотеката да остане след зареждането в паметта и да може да се извикват нейните функции, то тя трябва да върне значението TRUE:

MyDLL.cpp

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

int WINAPI DllMain(HINSTANCE hInstance, DWORD fdReason, PVOID
pvReserved)
{
   return TRUE;
}
EXPORT int CALLBACK MyFunction(char *str)
{
   MessageBox(NULL,str,"Function from DLL",MB_OK);
   return 1;  }
```

След транслация и свързващо редактиране на тези файлове се появяват два файла - **MyDLL.dll** (динамичната библиотека) и **MyDLL.lib** (нейната библиотека за импорта).

Пример за неявно включване на DLL в приложение

Код на просто приложение, което използва функцията MyFunction от библиотеката MyDLL.dll:

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

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
      LPSTR lpCmdLine, int nCmdShow)
{
   int iCode=MyFunction("Hello");
   return 0;}
```

Тя е нормална програма за Windows. Преди извикването на MyFunction от DLL-библиотеката то е включен и хедърния файл на библиотеки MyDLL.h. Така че в

процеса на свързаващо редактиране към него се включва и библиотеката на импорта MyDLL.lib (процес на неявно включване на DLL към изпълняем модул). Кодът на функцията MyFunction не се включва в файл MyApp.exe. В него просто има препратка към файл MyDLL.dll и към функцията MyFunction, която е в него. Файлът MyApp.exe изисква пускането и на файла MyDLL.dll.
Хедърният файл MyDLL.h е включен във файл с изходния текст на програмата MyApp.cpp точно така както и файл windows.h. Включването на библиотеката на импорта MyDLL.lib при свързването е аналогично на включването на всички библиотеки на импорта на Windows. Когато програмата MyApp.exe работи, тя се се свързва с библиотеката MyDLL.dll точно така както всички стандартни динамични включвания на библиотеките на Windows.

Пример за динамично зареждащо DLL приложение

Просто приложение, което използва функцията MyFunction от библиотеката MyDLL.dll, използвайки динамично зареждане на библиотеката.

```
#include <windows.h>
typedef int (WINAPI *PFN_MyFunction)(char *);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
        LPSTR lpCmdLine, int nCmdShow)
{
    HINSTANCE hMyDll;
    if((hMyDll=LoadLibrary("MyDLL"))==NULL) return 1;

    PFN_MyFunction pfnMyFunction;

pfnMyFunction=(PFN_MyFunction)GetProcAddress(hMyDll,"MyFunction");
    int iCode=(*pfnMyFunction)("Hello");

    FreeLibrary(hMyDll);
    return 0;
}
```

Създаване на DLL

Най-лесно е да създадем нов проект DLL с помощта на AppWizard, който автоматично ще изпълни операциите. За прости DLL, е необходимо да се избере тип на проекта Win32 Dynamic-Link Library. На новият проект ще бъдат присвоени всички необходими параметри за създаването на DLL библиотека. Файловете с входните текстове се добавят ръчно.
Ако се планира да се използват възможностите на MFC, или да се създадат сървърни приложения OLE, то по добре да се избере за тип на проекта MFC AppWizard (dll). В този случай в проекта трябва да бъдат добавени на MFC библиотеките и файловете както и входни текстове – реализации в библиотеката DLL на обект на класа на приложението, производен от **CWinApp**.

Понякога отначало се създава проект от типа MFC AppWizard (dll) в качеството на тестово приложение, а след това – да се въведат в DLL нейните съставни части.

Функция DllMain

Повечет блиотеки DLL – са просто колекции от практически независими една от друга функции, експортирани в приложенията и използвани в тях. Освен функции, предназначени за експортиране, във всяка DLL библиотека има и функция **DllMain**. Тази функция е предназначена за инициализация и изпразване на DLL.

Структура на най-проста функция DllMain може да изглежда така:

```
BOOL WINAPI DllMain (HANDLE hInst,DWORD dwReason, LPVOID
lpReserved)
  {
    BOOL bAllWentWell=TRUE;
    switch (dwReason)
    {
      case DLL_PROCESS_ATTACH:    // Инициализация на процеса
        break;
      case DLL_THREAD_ATTACH:     // Инициализация на потока
        break;
      case DLL_THREAD_DETACH:    // Изчистване структурите на потока
        break;
      case DLL_PROCESS_DETACH:   //Изчистване структурите на процесса
        break;
    }
    if(bAllWentWell)     return TRUE;
    else           return FALSE;
  }
```

Функция **DllMain** се извиква в няколко случаи. Причината за нейното извикване се определя от параметъра **dwReason**, който може да приема едно от следващите значения.
При първото зареждане на библиотеката DLL процесът извиква функцията **DllMain** с **dwReason**, равно на DLL_PROCESS_ATTACH. Всеки път при създаване в процеса на нов поток **DllMain** се извиква с **dwReason**, равен на DLL_THREAD_ATTACH (освен при първия поток, когато dwReason е DLL_PROCESS_ATTACH).
В края на работата на процеса с DLL функция **DllMain** се извиква с параметри dwReason, равен на DLL_PROCESS_DETACH. При унищожаване на потока (освен първия) dwReason ще е равен на DLL_THREAD_DETACH.
Всички операции по инициализация и изчистване за процесите и потоците е необходимо да се изпълнят на основата на значението dwReason.

Инициализация на процесите обикновено се ограничава до разделяемите ресурси, съвместно използвани от потоците, в частност зареждането на разделяемите файлове и инициализация на библиотеките. Инициализация на потоците се прилага за настройка на режимите, свойствени само на даден поток, например за инициализация на локална памет.
В състава на DLL могат да влязат ресурси, които не принадлежат на извикващото тази библиотека приложение. Ако функциите на DLL работят с ресурси на DLL, то е полезно да се съхранят някъде в свободно място на дескриптор hInst и да се използват при при зареждането на ресурси от DLL.

Указателят lpReserved е резервиран за вътрешно използване на Windows. Ако библиотеката DLL е била заредена динамично , то той ще е равен на NULL. При статическо зареждане този указател ще бъде ненулев.
В случай на успешно завършване на функция **DllMain** трябва да върне TRUE. В случай на възникване на грешка връща FALSE, и по-нататъшни действия се прекратяват.

Експортиране на функций от DLL

За да може приложението да се обръща към функциите на динамичната библиотека, за всяка от тях трябва да има ред в таблицата на експортируемите функции на DLL. Има два способа да се запише тази функция в тази таблица на етапа на компилация

Метод __declspec (dllexport)

Може да се експортират функции от DLL, поставяйки в началото на нейното описание **модификатора __declspec (dllexport)** .
Освен това, в състава на MFC влизат няколко макроса, определящи __declspec (dllexport), в това число AFX_CLASS_EXPORT, AFX_DATA_EXPORT и AFX_API_EXPORT.
Методът __declspec се прилага не така често, както втория метод, работещ с файловете, определени с модула (.def), и позволява по-добре да се управлява процеса на експортиране.

Файлове за определяне на модула

Синтаксисът на файловете расширение **.def** в Visual C++ е достатъчно праволинеен, заради това, че няма сложните параметри. Файлът .def съдържат име и описание на библиотеката, а така също и списък на експортируемите функции:

MyDLL.def

```
LIBRARY        "MyDLL"
   DESCRIPTION    'MyDLL - пример DLL-библиотеки'

EXPORTS
   MyFunction     @1
```

Я реда за експортни фунции може да се укаже нейния пореден номер, поставяйки пред него символа @. Този номер после може да се използва при обръщане към **GetProcAddress** (). Компилаторът присвоява номер на всички експортирани обекти. Този процес е непредсказуем, т.к тези номера не се присвояват явно.
В редовете на експорта може да се използва и параметър NONAME. Той забранява на компилатора да включва името на функцията в таблицата за експортиране на DLL:

```
MyFunction     @1 NONAME
```

Понякога с тази забрана може да се икономиса много място във файла DLL. Приложенията, използващи библиотеката за импортиране за неявного включване на DLL, няма да "забележат" разликата, т.к. при неявно включване поредните номера се използват автоматично. В приложенията, зареждащи библиотеките DLL динамично, се налага да се предават в **GetProcAddress** поредния номер, а не името на функцията.
При използване на външен def-файл описание на експортируемите функции от DLL-библиотеката може да не стане така:

```
#define EXPORT extern "C" __declspec (dllexport)
EXPORT int CALLBACK MyFunction(char *str);
```

а така:

```
extern "C" int CALLBACK MyFunction(char *str);
```

Цялостен пример за DLL библиотека и динамичното и извикване
DLL модули: главна подпрограма

```
BOOL WINAPI DllMain(HINSTANCE hinstDLL,DWORD fdwReason,LPVOID
lpvReserved)
{
switch (fdwReason)
{
case DLL_PROCESS_ATTACH:
/* Програмен код при натоварване на Dll*/
break;
case DLL_PROCESS_DETACH:
/* Програмен код при разтоварване на Dll */
break;
case DLL_THREAD_ATTACH:
/* Програмен код при старт на нишка */
break;
case DLL_THREAD_DETACH:
/* Програмен код при стоп на нишка */
break;
}
/* Кодът се връща при успешно натоварване на Dll */
return TRUE;
}
/*  една или повече функции */
__declspec (dllexport) DWORD Function1(.... ){}
```

```
__declspec (dllexport) DWORD Function2(.... ){
```

DLL модули: програмно тяло

```
/* Събиране на 2 цели числа */
_declspec (dllexport) long AddNumbers(long a, long b)
    {return((long)(a+b));}

/* Средна стойност на масив от числа */
_declspec (dllexport) long avg_num(float *a, long size, float *avg)
    { float sum=0;
    if(a != NULL){
    for(int i=0;i < size; i++)
    sum = sum + a[i]; }
    else
    return (1);
    *avg = sum / size;
    return (0); }

/* Преброява броя на целите числа в низ */
_declspec (dllexport) unsigned int numIntegers (char *inputString)
    { int lastDigit = 0;
    int numberOfNumbers = 0;
    int stringSize;
    stringSize = strlen(inputString);
    for(int i = 0; i < stringSize; i++)
    {
    if (!lastDigit && isdigit(inputString[i]))
    numberOfNumbers++;
    lastDigit = isdigit(inputString[i]);
    }
    return numberOfNumbers; }
```

DLL модули: натоварване на DLL и извикване на функция

```
#include <windows.h>
#include <stdio.h>
// DLL сигнатура на функцията
typedef double (*importFunction)(double, double);
int main(int argc, char **argv)
{
    importFunction AddNumbers;
    double result;
    // Зареждане на DLL файла
    HINSTANCE hinstLib = LoadLibrary(TEXT("Example.dll"));
    if (hinstLib == NULL) {
    printf("ERROR: unable to load DLL\n");
    return 1; }
    // Получаване на указател към функцията
    addNumbers = (importFunction)GetProcAddress(hinstLib, "AddNumbers");
    if (addNumbers == NULL) {
```

```
    printf("ERROR: unable to find DLL function\n");
    FreeLibrary(hinstLib);
    return 1; }
    // Извикване на функцията
    result = AddNumbers(1, 2);
    // Разтоварване на DLL модула
    FreeLibrary(hinstLib);
    // Показване на резултата
    printf("The result was: %f\n", result);
    return 0; }
```

Експортиране на класове

Създаването на .def-файл за експортиране даже в простите класове от динамичната библиотека може да се укаже много сложно. Налага се явно да се експортира всяка функция, която може да се използва от външно приложение. Ако се разглеждат на реализиран в клас файл ще се забележи следното.. Оказва се, че има неявни конструктори и деструктори, функции, обявени в макросите на MFC, в частност _DECLARE_MESSAGE_MAP, а така също и такива написани от програмиста.
Могат да се експортират тези функции поотделно по следния прост способ. Ако в обявяването на класа сме се възползвали от макромодификатора AFX_CLASS_EXPORT, компилаторът сам ще се погрижи за експортирането на необходимите функции, позволяващи приложението да използва клас, съдържащ се в DLL.

В Win32 библиотеката DLL се разполага в област от паметта на зареждащия го процес.На всеки процес се предоставя отделно копие на „глобалната" памет на DLL, което се инициализира всеки път, когатп се зарежда нов процес. Това означава, че динамичната библиотека неможе да се използва съвместнося совместно в обща памет.
С редица манипулации над сегментите на данните на DLL, може да се създаде обща област в паметта за всички процеси, използващи дадена библиотека.
Да допуснем, че масив от цели числа трябва да се използва от всички съвместни процеси, зареждащи дадената DLL. То тогава може да използваме следния програмен код:

```
#pragma data_seg(".myseg")
int sharedInts[10] ;
   // други промнливи за общо ползване
#pragma data_seg()
#pragma comment(lib, "msvcrt" "-SECTION:.myseg,rws");
```

Всички променливи между директивите #pragma data_seg(), се разполагат в сегмента **.myseg.** Директивата #pragma comment () - не е моментар. Тя дава указание указание за библиотеката на изпълняващата система.

Лекция 11 ActiveX технологии

Технологии ActiveX

- Клиентска технология ActiveX (Active Desktop)
- Сървърна технология ActiveX (Active Server)

ActiveX е технология на Microsoft, предназначена за написване на мрежови приложения. Тя предоставя на програмистите набор от стандартни библиотеки, които значително облекчават процеса на кодирането. Ако преди за написване на програми се използваха механизмите OLE (OLE Automation, OLE Documents, OLE Control,…), базирани на компонентния обектен модел (COM – Component Object Model), то сега библиотеките OLE са преписани така, че да осигуряват функционалност, достатъчна за написване на мрежови приложения. По този начин, сега при писането на програми се използва DCOM (DistributedComponent Object Model) – разпределен компонентен обектов модел, а него го осъществяват библиотеките ActiveX, които по обем се оказват много по-малки от библиотеката OLE, а по скорост са по бързи. Запазила се е и съвместимостта – всяка програмна компонента на OLE ще работи с библиотеките на ActiveX.

Доколкото най- динамично развиващото се направление в компютърната индустрия е Интернет, то именно тук могат най-естествено да намерят своето място програмите, написани с използването на технологията ActiveX. Не случайно в последно време понятията ActiveX и Интернет често се срещат заедно. В същото време технологията ActiveX има значително по-универсална област на приложение.

Стандартът ActiveX позволява на програмните компоненти да си взаимодействат по мрежата независимо от езика на програмиране, на който са написани. С помощта на ActiveX могат да се „оживят“ Web-страниците с ефектите на мултимедията, интерактивните обекти или сложните приложения.ActiveX осигурява известен „скрепяващ разтвор“, с помощта на който отделните програмни компоненти на различните компютри „се слепят“ в една разпределена система.

ActiveX включва клиентска и сървърна части, а така също и библиотеки за разработващия:

- **Програмни елементи ActiveX** – това са компоненти, работещи на компютъра на клиента, но зареждани първоначално от сървъра. С тяхна помощ може да се демонстрира разнородна информация, включваюа видео и звук без пускане на допълнителни програми. Освен това, тези програмни компоненти могат да се използват в приложения, написани на всякакви популярни езици за програмиране, включително на Java (Visual J++), Visual Basic, Visual C++.
- **Active Scripting** поддържа всеки популярен макроезик, включително Visual Basic Script и JScript (реализация на фирмата Microsoft на езика на сценарии JavaScript). Макроезиците могат да се използват за обединяване на една

страница на няколко програмни елемента ActiveX или Java,осигурявайки взаимодействието между тях.

- **Документите ActiveX** дават възможност да се отварят и обработват в прозореца Microsoft Internet Explorer документ от всякакъв формат (например, файл Microsoft Excel или Word).
- **Виртуалната машина Java** позволява всяка програма за преглед на Интернет, поддържаща технологията ActiveX (например, Internet Explorer 3.0) да изпълнява програмни компоненти на Java и да осигурява тяхното взаимодействие с програмни компоненти на ActiveX.
- **ActiveX Server Framework** осигурява сървърни функции на ActiveX, включително поддържане на безопасни съединения, достъп до бази данни и др.
- **Средствата за разработване** позволяват използването на познатите системи за програмиране на Microsoft и на други фирми за създаване на компоненти на ActiveX. Към тях се отнасят Visual Basic, Visual C++, Macromedia Shockwave, Adobe Photoshop, Borland Delphi, средствата за програмиране Sybase и други.

Основни предимства на използването на технологията ActiveX:

- **Бързо изписване на програмния код.** Програмирането на мрежови взаимодействия заприличва много на програмиране за отделен компютър.
- **Отвряемост и мобилност.** Спецификациите на технологията наскоро беше предадена в Open Group като основа на отворения стандарт. Освен това, Microsoft съвместно с фирмите Metrowerks и Bristol завършват реализацията на технологията ActiveX за платформите Macintosh и UNIX.
- **Възможност за писане на приложения с използване на познатите средства за разработка.** Програмните елементи на ActiveX могат да бъдат създадени с помощта на Visual Basic, Visual C++, Borland Delphi, Borland C++, всякакви средстваза разработване на Java.
- **Голямо количество вече съществуващи програмни елементи на ActiveX,** които могат безплатно да се прилагат на сървърите и в приложенията на независимите разработчици. Освен това, почти всеки програмен компонент на OLE е съвместим с технологиите ActiveX и може да се прилага без промени в мрежовите приложения.
- **Стандартност.** Технологията ActiveX се основава на широко използваните стандарти Internet (TCP/IP, HTML, Java) от една страна и на стандартите, въведени междувременно от Microsoft и необходими за запазване на совместимостта на (COM, OLE).

Клиентската технология ActiveX (Active Desktop)

ActiveX се реализира на машината – клиент с помощта на библиотеките, доставяни заедно с Internet Explorer. По-нататък тези библиотеки ще се допълват и преписват за оптимално предаване по мрежата на данните на мултимедията.

Програмните компоненти на ActiveX могат да бъдат инсталирани автоматично на компютъра на потребителя по мрежи с отдалечен сървър, при което ще бъде въведен код, подходящ за конкретната платформа на клиента, пък била тя и Macintosh, Windows или Unix. Разработващият Web-страниците може или сам да запрограмира елементите

на ActiveX, като използва популярните езици за програмиране Visual C++, Visual Basic или Java, или да използва съществуващите. Примери за готови програмни елементи можете да намерите на адрес http://www.microsoft.com/activeplatform/default.asp.

Използвайки езиците на сценариите на ActiveX програмиститемогат да осигурят взаимодействието на различните елементи на ActiveX, Java, други програми на компютъра на клиента и различни части на самия Internet Explorer. Например, програмният елемент на синхронизацията може да обновява WEB-страницата през определени периоди от време.Може, също така, периодически да се пуска програмен елемент, привличащ вниманието на потребителя. Има реализации на Visual Basic Scripting Edition, които са подмножество на Visual Basic и JScript. Освен това разработващият може да напише интерпретатор на собствения език на сценариите и да го добави в системата.

С ActiveX Documents е запознат всеки, който е работил със съставни документи. С помощта на Internet Explorer може да се работи, например, с таблиците на Microsoft Excel и с файловете на други офисни приложения. Това прави програмата на прегледа универсално средство, способно не само да отразява файлове във формат HTML и да осъществява преходи по изпращането, но и поддържащ работата с документи на всякакви приложения и дори да пуска програми.

Сървърна технология ActiveX (Active Server)

Сървърната част на технологията ActiveX е реализирана с помощта на Microsoft Internet Information Server С помощта на ActiveX могат да се пишат програми на езиците на сценариите, които се изпълняват на сървъра. Ако преди на разработващите им се налагаше да използват такива средства, като Microsoft Visual C++ за писането на динамично зареждащи се библиотеки, използващи специалните повиквания Internet Server API, то сега е възможно да се пишат приложения на езика на сценариите. Това съществено опростява разработката, съкращава времето за писане на програми и минимизира разходите. Програмите, базирани на технологията Active Server, са в пъти по-производителни от програмите, базирани на Common Gateway interface (CGI). Това се постига с оптимизация на процесите на ActiveX на сървъра, отчитащ архитектурата Windows.

С помощта на езиците на сценариите, на сървъра може да се осъществява достъп до системите за управление на БД, поддържащи стандарта ODBC, и да се използва механизма на транзакциите.

Тъй като подходът към използване на технологиите ActiveX на сървъра е стандартизиран, програмистите могат не само да разработват приложения, способни да се изпълняват на сървърите, но и да реализират свои схеми на взаимовръзка на сървърните приложения и сервизите, да създават собствени интерпретатори на сървърните езици на сценариите. За целта се изисква предварително придобиване на лиенз от Open Group.

Лекция 12

Приложения с БД

Създаването на приложения, които използват база данни за C++ става със следната процедура:

1. Създава се нов проект от тип CLR, празен проект.

2. Добавя се от Project->AddNewItem->UI->Form

3. Добавя се в текста на MyForm.cpp: #include "MyForm.h"

в MyForm.cpp .

4. Добавяне на елементи – контроли за работа с ДБ : OleDbConnection, OleDbAdapter

5. Осъществяване на връзката с БД чрез Tools-> Connection

6. След успешна връзка средата предоставя възможност за :

създаване и редакция на таблици, представяния, процедури, функции

7. Операции с БД.

Open Database Connectivity (ODBC) е стандартен интерфейс за приложно програмиране (API) за достъп до системи за управление на бази данни DBMS (СУБД). ODBC целят да е независим от системите за бази данни и операционните системи. Заявките написани на ODBC могат да бъдат пренесени към други платформи, както от страна на клиента така и на сървъра, с няколко промени в кода. За него има написан специален драйвер, които осигурява транслация между приложението и СУБД.

#include <windows.h>

.....

```
#include <stdio.h>
#include <vector>
#include <string>
#include <sql.h>
#include <sqlext.h>
#include <odbcinst.h>
#include <odbcss.h>

#define MAX_LOADSTRING 100
#define NAME_LEN 50

using namespace std;

typedef vector<string> VSTR;
VSTR vstr;
VSTR::iterator iter;
```

****

```
// add in WinMain
int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
      // TODO: Place code here.
      MSG msg;
      HACCEL hAccelTable;

      // Initialize global strings
      LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
      LoadString(hInstance, IDC_DB, szWindowClass, MAX_LOADSTRING);

      MyRegisterClass(hInstance);

      // Fill vector
      SQLHENV     henv;
      SQLHDBC     hdbc;
      SQLHSTMT    hstmt;
      SQLRETURN   retcode;

      SQLCHAR      szName[NAME_LEN];
      SQLINTEGER   sID, sGY, cbName, cbID, cbGY;
      char res[100];
// заделя среда, връзка и описател на хендъл
      SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
// задава атрибути на връзката
      SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);

      SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
//
      SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (void*)5, 0);
// установяване на връзка
SQLConnect(hdbc, (SQLCHAR*) "stu", SQL_NTS,
            (SQLCHAR*) "", SQL_NTS, (SQLCHAR*) "", SQL_NTS);

SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

      retcode = SQLExecDirect(hstmt, (SQLCHAR*)"SELECT * FROM Student",
SQL_NTS);
      if(retcode == SQL_ERROR)
            int i=0;

      SQLBindCol(hstmt, 1, SQL_C_ULONG, &sID, 0, &cbID);
      SQLBindCol(hstmt, 2, SQL_C_CHAR, szName, NAME_LEN, &cbName);
      SQLBindCol(hstmt, 3, SQL_C_ULONG, &sGY, 0, &cbGY);

      while(TRUE)
      {
            retcode = SQLFetch(hstmt);
            if(retcode == SQL_ERROR || retcode == SQL_NO_DATA)
                  break;
            sprintf(res, "%ld %s %d", sID, szName, sGY);
            vstr.push_back(res);
      }

      SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
      SQLDisconnect(hdbc);
```

```
      SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
      SQLFreeHandle(SQL_HANDLE_ENV, henv);
      // End fill vector

      // Perform application initialization:

                  ********
// add in WndProc

      PAINTSTRUCT ps;

      TEXTMETRIC tm;
      int nLineH;
      POINT pT;

      switch (message)

                  ********
// add in WndProc

            case WM_PAINT:
                  hdc = BeginPaint(hWnd, &ps);

                  GetTextMetrics(hdc, &tm);
                  nLineH = tm.tmHeight+tm.tmExternalLeading;
                  pT.x = 0; pT.y = 0;

                  for(iter = vstr.begin(); iter != vstr.end(); iter++)
                  {
                        TextOut(hdc, pT.x, pT.y, (*iter).c_str(),
(*iter).length());
                        pT.y += nLineH;
                  }

                  EndPaint(hWnd, &ps);
                  break;
```

Създаване на COM компонента на Visual C++

Външните компоненти се използват както за сървъра (Windows и Linux, 32 или 64 бит), така и за "тънки" и web -клиенти.

Създаване на външна компонента се изпънява в програмите на Microsoft Visual Studio. Компонентата трябва да се включва към шаблон за сървър Windows 32 бита приложение.

Най-просто създаването на MFC компонента е да се заменят няколко функции в готов шаблон на външна компонента:

1. ДА се присвои на класа **C1CGetImageFragment** ново име, например **MyAddIn**.
2. ДА се преименуват файловете, например **1CGetImageFragment.h** на **MyAddIn.h** и **1CGetImageFragment.cpp** на **MyAddIn.cpp**.
3. В файла **MyAddIn.h** в изчислението **enum Methods** да се зададе друго име например Version.

```
4.    {
5.    eVersion = 0, //
6.    eGetImageFragment, //
7.    eMethLast // Always last
      };
```

8. В массива реда **g_MethodNames** и **g_MethodNamesRu** да се укаже името на вашите функции.
9. В реда **g_kClassNames** да се укаже името на вашия клас, например **MyAddIn**.
10. В функцията **GetNParams** да се укажат аргументите на вашите методи.
11. При необходимост в функцията **GetParamDefValue** да се укажат аргументи по подразбиране за вашите методи.
12. В функцията **HasRetVal** да се укаже връща ли вашата функция значение. Например:

```
13.    case eGetImageFragment:
14.    return true;
```

15. В функцията **CallAsFunc** да се добави код на една или няколко нови функции.

Следва: да се компилира компонентата в Release-вариант и да се получи файл в вида dll-библиотека, например **MyAddIn.dll**. Той трябва да е заедно с файла **Manifest.xml**.

Описание на създаването на проста COM компонента на Visual C++, която няма да прави нищо полезно, а просто ще послужи за създаване на COM-и, които ще бъдат по-функционални.

И така, отваря се Visual C++, в меню New изберете ATL COM AppWizard, след това напишете в полето Project Name: MyCom и натиснете Ok. В следващ прозрци на Wizard може да се избере и типа DLL.

След създаването на проекта се добавя в него COM-обект, като натиснете Insert в главното меню и изберете New ATL Object...

Изберете категорията Objects и обекта Simple Object. Сега виждате диалоговия прозорец, кликнете върху Names, в полето Short Name въведете името на компонентата, която възнамерявате да създадете, нека това да е MyObject, забележете, че всички останали полета се запълват автоматично, препоръчително е така и да остане.

Ако искате, можете да измените полето type, това е просто описание на COM-а, нека да въведем My first Class. Сега кликнете върху Attributes, изберете Single threading model, Custom interface и No Aggregation. Това е всичко, вие създадохте компонента, а сега трябва да я направите работеща. В прозореца ClassView вие виждате създадения от вас клас CMyObject и интерфейса IMyObject, той е необходим за създаване на библиотека от типове.

Изберете интерфейс IMyObject в ClassView, кликнете върху него с десния бутон на мишката и от контекстното меню изберете Add Method...

Сега виждате пред себе си диалоговия прозорец Add Method to Interface, в полето Method Name въведете ShowMessageBox, а в полето parameters въведете:[in] const BSTR StringToWrite, [out,retval] long *Result. Този ред: [in] показва, че StringToWrite се въвежда във функцията, а [out] - че този параметър се възвръща, [retval] означава, че този параметър ще се връща от всяка функция; това е необходимо за да може компонентата да работи, например, в средата Visual Basic, т.к VB не поддържа типа данни HRESULT, който връща тази функция. Вместо да връща даденото от типа HRESULT, във VB този метод ще върне даденото от типа long. Методът е създаден. В нашия пример той ще показва на потребителя на прозореца съобщение с текст, намиращ се в променливата StringToWrite.

Нека сега да създадем свойство от редов тип, което ще отговаря за заглавието на прозореца на съобщението, може би ще е крайно в свойството ShowMessageBox да добавим още един [in] параметър,който би отговарял за това, но сега нашата цел е да разработим демонстрационна компонента и затова ще създадем свойството. Кликнете отново в интерфейса IMyObject с десния бутон на мишката и изберете Add Property... В поле Property Type изберете BSTR, а в поле Property Name въведете Caption и кликнете върху Ok. В ClassView изберете CMyObject->IMyObject и ще видите 2 функции: get_Caption и put_Caption. Функция put_Caption се извиква, когато присвоявате на свойството Caption ново значение, а get_Caption – когато получавате. В интерфейса тези функции ги няма; сега можете да компилирате проекта и да отидете във Visual Basic.

Изберете меню Project->References, намерете в списъка MyCom 1.0 Type Library, отбележете го с √ и кликнете на Ok. В модула напишете:

```
Dim MyObj As New MyObject

Private Sub Form_Load()
     MyObj.
End Sub
```

Пред вас се отваря списък с методи и свойства, както виждате тук има само 1 метод ShowMessageBox и само 1 свойство Caption. Нито едното, нито другото работи, защото още не сме реализирали метода ShowMessageBox и функциите put_Caption и get_Caption. Да ги накараме да работят! Отначало ще реализираме метода ShowMessageBox. В ClassView изберете CMyObject->IMyObject->ShowMessageBox и напишете тук следния код:

```
_bstr_t temp(StringToWrite);
_bstr_t caption(m_Caption);
*Result=MessageBox(NULL,temp,caption,MB_YESNO|MB_ICONINFORMATION);
temp.~_bstr_t();
caption.~_bstr_t();
return S_OK;
```

Тук ще видите променливата m_Caption, която ни предстои да създадем в бъдеще за връзка на свойството Caption и компонентата, а вероятно непознатия клас _bstr_t. _bstr_t предоставя полезни оператори и методи за работа с типа BSTR, но за да го използвате, трябва да включите header comdef.h, отворете MyObject.h и след редовете #include "resource.h" // main symbols добавете #include "comdef.h". В този код MessageBox ще върне или IDYES, или IDNO, в зависимост от това кой бутон ще натисне потребителя, и това значение ще възвръща функция, т.к. сме написали *Result=MessageBox..., а Result е възвръщащ се параметър. Възможно е типа long за Result да е бил избран грешно, т.к MessageBox връща числата от 1 до 9, но, както вече казах, тази компонента е просто тест. Сега остана да се реализират функциите put_Caption и get_Caption. За да направите това, трябва първо да добавите в клас CMyObject защитената променлива m_Caption от типа BSTR, нея също я няма в интерфейса, но тя ще се използва за да съхрани това значение, което потребителят е присвоил на свойството Caption. Отворете файла MyObject.h и след редовете:

```
public:
    STDMETHOD(get_Caption)(/*[out, retval]*/ BSTR *pVal);
    STDMETHOD(put_Caption)(/*[in]*/ BSTR newVal);
    STDMETHOD(ShowMessageBox)(/*[in]*/ const BSTR StringToWrite,
/*[out,retval]*/
long *Result);
```

напишете:

```
protected:
    BSTR m_Caption;
```

Тук точно, в конструктора на класа напишете:

```
_bstr_t temp("Just a test!");
m_Caption=temp.copy();
temp.~_bstr_t();
```

Сега променливата m_Caption е инициализирана, с премълчаване в нея ще се намира израза"Just a test!". А сега, можете на края да запълните функциите get_Caption и put_Caption. Отворете файл MyObject.cpp, намерете там функцията get_Caption и в нея напишете:

```
*pVal=m_Caption;

return S_OK;
```

Сега намерете функцията put_Caption и в нея напишете:

```
m_Caption=newVal;

return S_OK;
```

Следва компилиране на компонентата.

Server (dll, simple)

```
/******************** Server.cpp **********************/

#include "stdafx.h"
#include "Numbers.h"
#include <stdio.h>

const char* SCLSID_Numbers = "{B58DF065-EAD9-11d7-BB81-
000475BB5B75}";
HMODULE g_hModule;
long g_locks;
CNumbersF gNumF;

BOOL APIENTRY DllMain(HANDLE hModule, DWORD
ul_reason_for_call, LPVOID lpReserved)
{
     if(ul_reason_for_call == DLL_PROCESS_ATTACH)
     {
          g_hModule = (HINSTANCE)hModule;
     }
     g_locks = 0;
     return TRUE;
}

STDAPI DllCanUnloadNow()
{
     return g_locks == 0 ? S_OK : S_FALSE;
}

STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid, void**
ppv)
{
     *ppv = NULL;
     if (rclsid == CLSID_Numbers)
          return gNumF.QueryInterface(riid, ppv);
     return CLASS_E_CLASSNOTAVAILABLE;
}

BOOL HelperWriteKey(const char *lpSubKey, const char*
val_name, const char* szData)
{
```

```
     HKEY hk;
     RegCreateKey(HKEY_CLASSES_ROOT, lpSubKey, &hk);
     RegSetValueEx(hk, val_name, 0, REG_SZ, (const BYTE
*)szData, strlen(szData));
     RegCloseKey(hk);
     return TRUE;
}

STDAPI DllRegisterServer()
{
     char szSubKey[MAX_PATH], szBuff[MAX_PATH];
     sprintf(szSubKey, "CLSID\\%s", SCLSID_Numbers);
     HelperWriteKey(szSubKey, NULL, "Numbers Component");
     sprintf(szSubKey, "CLSID\\%s\\InprocServer32",
SCLSID_Numbers);
     GetModuleFileName(g_hModule, szBuff, sizeof(szBuff));
     HelperWriteKey(szSubKey, NULL, szBuff);
     sprintf(szSubKey, "CLSID\\%s\\ProgId", SCLSID_Numbers);
     HelperWriteKey(szSubKey, NULL, "Server.Numbers");
     HelperWriteKey("Server.Numbers", NULL, "Numbers
Component");
     sprintf(szSubKey, "%s\\CLSID", "Server.Numbers");
     HelperWriteKey(szSubKey, NULL, SCLSID_Numbers);
     return S_OK;
}


STDAPI DllUnregisterServer()
{
     char szSubKey[MAX_PATH];
     sprintf(szSubKey, "CLSID\\%s\\InprocServer32",
SCLSID_Numbers);
     RegDeleteKey(HKEY_CLASSES_ROOT, szSubKey);
     sprintf(szSubKey, "CLSID\\%s\\ProgId", SCLSID_Numbers);
     RegDeleteKey(HKEY_CLASSES_ROOT, szSubKey);
     sprintf(szSubKey, "CLSID\\%s", SCLSID_Numbers);
     RegDeleteKey(HKEY_CLASSES_ROOT, szSubKey);
     sprintf(szSubKey, "%s\\bCLSID", "Server.Numbers");
     RegDeleteKey(HKEY_CLASSES_ROOT, szSubKey);
     RegDeleteKey(HKEY_CLASSES_ROOT, "Server.Numbers");
     return S_OK;
}


/******************  Server.def **********************/

DESCRIPTION    "Numbers Component"

EXPORTS
          DllGetClassObject        PRIVATE
          DllCanUnloadNow          PRIVATE
```

```
          DllRegisterServer       PRIVATE
          DllUnregisterServer     PRIVATE
```

```
/*******************  Numbers.idl **********************/

import "unknwn.idl";

[ local, uuid(B58DF063-EAD9-11d7-BB81-000475BB5B75) ]
interface INumbers: IUnknown
{
     HRESULT GetVal42(INT* val);
};

[ uuid(B58DF064-EAD9-11d7-BB81-000475BB5B75),
helpstring("Server 1.0 Type Library") ]
library SERVERLib
{
     importlib("stdole2.tlb");

     [ uuid(B58DF065-EAD9-11d7-BB81-000475BB5B75) ]
     coclass Numbers
     {
          interface INumbers;
     }
};

// Project->Settings->Numbers.idl->MIDL => uncheck-MkTypLib;
Output-Numbers.tlb; Output header-some.h
```

```
/*******************  Numbers.h **********************/

#ifndef _H_
#define _H_

#include <objbase.h>

extern long g_locks;

const GUID IID_INumbers            =
{0xB58DF063,0xEAD9,0x11d7,{0xBB,0x81,0x00,0x04,0x75,0xBB,0x5B,
0x75}};
const GUID LIBID_SERVERLib         =
{0xB58DF064,0xEAD9,0x11d7,{0xBB,0x81,0x00,0x04,0x75,0xBB,0x5B,
0x75}};
const GUID CLSID_Numbers      =
{0xB58DF065,0xEAD9,0x11d7,{0xBB,0x81,0x00,0x04,0x75,0xBB,0x5B,
0x75}};


interface INumbers: public IUnknown
{
```

```
public:
     virtual HRESULT __stdcall GetVal42(INT* val) = 0;
};

class CNumbers: public INumbers
{
public:
     CNumbers();
     virtual ~CNumbers();
     virtual ULONG __stdcall AddRef();
     virtual ULONG __stdcall Release();
     virtual HRESULT __stdcall QueryInterface(REFIID riid,
void** ppv);
     virtual HRESULT __stdcall GetVal42(int* val);
protected:
     long refcount;
};

class CNumbersF: public IClassFactory
{
public:
     virtual ULONG __stdcall AddRef()   { return 0; }
     virtual ULONG __stdcall Release()  { return 0; }
     virtual HRESULT __stdcall QueryInterface(REFIID riid,
void** ppv);
     virtual HRESULT __stdcall CreateInstance(IUnknown*
pUnkOuter, REFIID riid, void** ppv);
     virtual HRESULT __stdcall LockServer(BOOL bLock);
};
#endif

/*******************  Numbers.cpp **********************/

#include "stdafx.h"
#include "Numbers.h"

CNumbers::CNumbers()
{
     refcount = 0;
     g_locks++;
}

CNumbers::~CNumbers()
{
     g_locks--;
}

STDMETHODIMP_(ULONG) CNumbers::AddRef()
{
     return InterlockedIncrement(&refcount);
}
```

```
STDMETHODIMP_(ULONG) CNumbers::Release()
{
     if (InterlockedDecrement(&refcount) == 0)
     {
          delete this;
          return 0;
     }
     return refcount;
}

STDMETHODIMP CNumbers::QueryInterface(REFIID riid, void** ppv)
{
     if (riid == IID_IUnknown)
          *ppv = (IUnknown*)this;
     else if (riid == IID_INumbers)
          *ppv = (INumbers*)this;
     else
     {
          *ppv = NULL;
          return E_NOINTERFACE;
     }
     ((IUnknown*)*ppv)->AddRef();
     return S_OK;
}

STDMETHODIMP CNumbers::GetVal42(int* val)
{
     if(val == NULL)
          return E_POINTER;
     *val = 42;
     return S_OK;
}

STDMETHODIMP CNumbersF::QueryInterface(REFIID riid, void**
ppv)
{
     if (riid == IID_IUnknown)
          *ppv = (IUnknown*)this;
     else if (riid == IID_IClassFactory)
          *ppv = (IClassFactory*)this;
     else
     {
          ppv = NULL;
          return E_NOINTERFACE;
     }
     ((IUnknown*)*ppv)->AddRef();
     return S_OK;
}

STDMETHODIMP CNumbersF::CreateInstance(IUnknown* pUnkOuter,
```

```
REFIID riid, void** ppv)
{
     CNumbers* pNum;
     HRESULT hr;

     if (pUnkOuter)
          return CLASS_E_NOAGGREGATION;
     pNum = new CNumbers;
     if (!pNum)
          return E_OUTOFMEMORY;
     hr = pNum->QueryInterface(riid, ppv);
     if (FAILED(hr))
          delete pNum;
     return hr;
}

STDMETHODIMP CNumbersF::LockServer(BOOL bLock)
{
     if (bLock)
          InterlockedIncrement(&g_locks);
     else
          InterlockedDecrement(&g_locks);
     return S_OK;
}
```

Client (win32 exe, simple)

```
/*******************  Client.cpp **********************/

#include "stdafx.h"
#include <stdio.h>
#include <objbase.h>
#import "..\numbers.tlb"

const IID IID_INumbers =
{0xB58DF063,0xEAD9,0x11d7,{0xBB,0x81,0x00,0x04,0x75,0xBB,0x5B,
0x75}};
const CLSID CLSID_Numbers =
{0xB58DF065,0xEAD9,0x11d7,{0xBB,0x81,0x00,0x04,0x75,0xBB,0x5B,
0x75}};

SERVERLib::INumbers* pNum;
HRESULT hr;


int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
```

```
{
	hr = CoInitialize(NULL);
	if(hr == S_OK)
	{
		hr = CoCreateInstance(CLSID_Numbers, NULL,
CLSCTX_INPROC_SERVER, IID_INumbers, (void**)&pNum);
		if(SUCCEEDED(hr))
		{
			int val;
			pNum->GetVal42(&val);
			pNum->Release();
			char szBuf[100];
			sprintf(szBuf, "GetVal42 ret: %d", val);
			MessageBox(NULL, szBuf, "Ret", MB_OK);
		}
		CoUninitialize();
	}
	return 0;
}
```

Външните компоненти се използват както за сървъра (Windows и Linux, 32 или 64 бит), така и за "тънки" и web -клиенти.

Създаване на външна компонента се изпълнява в програмите на Microsoft Visual Studio. Компонентата трябва да се включва към шаблон за сървър Windows 32 бита приложение.

Най-просто е да се заменят няколко функции в готов шаблон на външна компонента:

16. ДА се присвои на класа **C1CGetImageFragment** ново име, например **MyAddIn**.
17. ДА се преименуват файловете, например **1CGetImageFragment.h** на **MyAddIn.h** и **1CGetImageFragment.cpp** на **MyAddIn.cpp**.
18. В файла **MyAddIn.h** в изчислението **enum Methods** да се зададе друго име например Version.

```
19.  {
20.  eVersion = 0, //
21.  eGetImageFragment, //
22.  eMethLast // Always last
     };
```

23. В массива реда **g_MethodNames** и **g_MethodNamesRu** да се укаже името на вашите функции.
24. В реда **g_kClassNames** да се укаже името на вашия клас, например **MyAddIn**.
25. В функцията **GetNParams** да се укажат аргументите на вашите методи.
26. При необходимост в функцията **GetParamDefValue** да се укажат аргументи по подразбиране за вашите методи.
27. В функцията **HasRetVal** да се укаже връща ли вашата функция значение. Например:

```
28.     case eGetImageFragment:
29.     return true;
```

30. В функцията **CallAsFunc** да се добави код на една или няколко нови функции.

Следва: да се компилира компонентата в Release-вариант и да се получи файл в вида dll-библиотека, например **MyAddIn.dll**. Той трябва да е заедно с файла **Manifest.xml**.