Архив статей wasm.ru. Том 2: COM, OOP, DirectX и OpenGL

Собрание русскоязычных статей сайта wasm.ru о программировании, ассемблере, реверс-инжиниринге и компьютерной безопасности. Том 2 из 11. Разделы: 3. COM и OOP; 4. DirectX/OpenGL. Все приложенные файлы находятся в wasm_files.zip.

3. COM и OOP

COM в Ассемблере - Часть I

Автор: Bill T., пер. Aquila

Дата: 2002-06-27

В этой статье будет расказано о том, как использовать COM-интерфейсы в ваших программах, написанных на ассемблере. Не будет обсуждаться, что такое COM и как он применяется, но как его можно использовать, программируя на ассемблере. Здесь будет затронуто только применение существующих интерфейсов, а не реализация своих собственных, это будет рассмотрено в другой статье.

О COM

Это краткое введение в основы COM.

Получить доступ к COM-объекту можно только через один или большее количество наборов связанных с ним функций. Эти наборы функций называются интерфейсами, а функции интерфейса называются методами. COM требует, чтобы существовал только один путь доступа к методам интерфейса - через указатель на интерфейс.

По терминологии COM, интерфейс - это "контракт", состоящий из группы связанных друг с другом прототипов функций, чье использование определено, а реализация - нет. Определение интерфейса задает функции интерфейса, называемые методами, типы возвращаемых ими значений, количество и типы их параметров, и что они должны делать. С интерфейсом не ассоциируется какая-то конкретная его реализация. Реализация интерфейса - это код, который предоставляет программист для выполнения действий, заданных определением интерфейса.

Экземпляр реализации интерфейса - это указатель на массив указателей на методы (таблица указателей, ссылающиеся на реализацию всех методов, указанных в интерфейсе). Любой код, у которого есть подобный указатель, может вызывать методы этого интерфейса.

Использование COM-объекта в Ассемблере

Доступ к COM-объекту осуществляется через указатель, который указывает на таблицу указателей на функции (эту таблицу еще называют таблицей виртуальных функций или vtable для краткости). Эта таблица содержит адреса каждого из методов объекта. Чтобы вызывать метод, вы косвенно вызываете его через эту таблицу указателей.

Здесь приводится пример интерфейса на C++, и как называются его методы:

```
interface IInterface
{
          HRESULT QueryInterface( REFIID iid, void ** ppvObject );
          ULONG AddRef();
          ULONG Release();
          Function1( INT param1, INT param2);
          Function2( INT param1 );
}
// вызываем метод Function1
```

```
pObject->Function1( 0, 0);
```

То же самое можно реализовать на ассемблере следующим образом:

```
; определяем интерфейс
     ; каждое из этих значений - это смещение в vtable
     QueryInterface          equ             0h
     AddRef                  equ             4h
Release          equ   8h
Function1    equ    0Ch
Function2    equ    10h

; вызываем метод Function1 на ассемблере
     ; вызываем метод, получая адрес таблицы виртуальных функций,
     ; а затем вызываем функцию через указатель, находящийся по
     ; нужному смещению в таблице
push   param2
push   param1
     mov     eax, pObject
push   eax
     mov     eax, [eax]
call   [eax + Function1]
```

Вы можете видеть, что это отличается от вызова обычной функции. Здесь pObject указывает на таблицу интерфейса. По смещению Function1 (0Ch) в этой таблице находится указатель на саму функцию, которую мы хотим вызвать.

Использование HRESULT

Возвращаемое значение функциями OLE API и методами является HRESULT. Это не хэндл чег-нибудь, а просто 32-х битное значение с несколькими полями. Части HRESULT показаны ниже.

HRESULT - это 32-х битное значение со следующей структурой.

```
 3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-+-+-+-+-+---------------------+-------------------------------+
|S|R|C|N|r|    Facility         |               Code            |
+-+-+-+-+-+---------------------+-------------------------------+

 S - Severity Bit
     Используется для того, чтобы сообщить, была ли функция выполнена
     успешно или нет.

     0 - Успех
     1 - Провал

     Так как этот бит фактически является битом знака 32-х битного значения,
     проверить, успешно была выполнена функция или нет, можно просто
     проверив его знак:

     call       ComFunction        ; вызываем функцию
     test       eax,eax            ; теперь проверяем возвращенное значение
     js         error              ; делаем переход, если установлен бит
                                   ; знака (произошла ошибка)
     ; успех, продолжаем выполнение программы

 R - зарезервированная часть кода facility.

 C - зарезервированная часть кода facility.

 N - зарезервированная часть кода facility.

 r - зарезервированная часть кода facility

 Facility - это код facility
```

```
    FACILITY_WINDOWS    = 8
    FACILITY_STORAGE    = 3
    FACILITY_RPC        = 1
    FACILITY_WIN32      = 7
    FACILITY_CONTROL    = 10
    FACILITY_NULL       = 0
    FACILITY_ITF        = 4
    FACILITY_DISPATCH   = 2

    Чтобы получить этот код:

    call       ComFunction    ; вызываем функцию
    shr        eax, 16        ; сдвигаем HRESULT вправо на 16 бит
    and        eax, 1FFFh     ; маскируем биты так, что остается только
                              ; код facility
    ; теперь eax содержит HRESULT'овский код facility

Code - код статуса facility

    Чтобы получить код статуса facility
    call       ComFunction             ; вызываем функцию
    and  eax, 0000FFFFh     ; обнуляем верхние 16 бит
    ; теперь eax содержит the HRESULT'овский код статуса facility
```

Использование COM в MASM

Если вы используете MASM для ассемблирования ваших программ, вы можете использовать некоторые из его возможностей, чтобы сделать вызов COM-функций очень простым. Используя invoke, вы можете сделать COM-вызовы почти такими же понятными как вызовы обычных функций, плюс вы можете добавить проверку типов для каждой функции.

Определение интерфейса:

```
    IInterface_Function1Proto     typedef proto :DWORD
    IInterface_Function2Proto     typedef proto :DWORD, :DWORD

    IInterface_Function1          typedef ptr IInterface_Function1Proto
    IInterface_Function2          typedef ptr IInterface_Function2Proto

    IInterface struct DWORD
          QueryInterface          IUnknown_QueryInterface         ?
          AddRef                  IUnknown_AddRef                 ?
          Release                 IUnknown_Release                ?
          Function1               IInterface_Function1            ?
          Function2               Interface_Function2             ?
    IInterface ends
```

Использование интерфейса для вызова COM-функций:

```
    mov     eax, pObject
    mov     eax, [eax]
    invoke  (IInterface [eax]).Function1, 0, 0
```

Как вы можете видеть, синтакс может выглядеть немного странно, но это дает возможность использовать имя функции вместо смещений, а заодно и проверку типов.

Программа-пример с использованием COM

Вот исходник, который написан так, чтобы быть максимально совместимым с любым ассемблером, который вы предпочтете (по крайней мере, чтобы вам не пришлось делать глобальных изменений).

Эта программа использует интерфейсы Windows Shell, чтобы отобразить содержиом Рабочего стола. Программа не закончена, но она показывает, как инициализировать библиотеку COM, деинициализировать и использовать. Я также покажу, как использовать shell-библиотека, чтобы получить папки и объекты, и выполнять над ними различные действия

```
.386
.model flat, stdcall

include windows.inc    ; подключает стандартных заголовочный файл
include shlobj.inc     ; этот заголовочный файл содержит константы и
                       ; определения shell'a

;-------------------------------------------------------
.data
        wMsg                    MSG     <?>
        g_hInstance             dd      ?
   g_pShellMalloc      dd ?

        pshf                    dd      ?       ; объект папки shell'a
        peidl                   dd      ?       ; объект списка id

        lvi                     LV_ITEM <?>
        iCount                  dd      ?
        strret                  STRRET  <?gt;
        shfi                    SHFILEINFO <?>
   ...

;-------------------------------------------------------
.code
; Entry Point
start:
    push    0h
    call    GetModuleHandle
    mov     g_hInstance,eax

    call    InitCommonControls

; Инициализируем библиотеку Component Object Model (COM)
    push    0
    call    CoInitialize
    test    eax,eax             ; ошибка, если MSB = 1
                                ; (MSB = бит знака)
    js      exit                ; js = переход, если установлен бит знака

; Получаем указатель на объект shell'a IMalloc и сохраняем его в глобальную
; переменную
    push    offset g_pShellMalloc
    call    SHGetMalloc
    cmp     eax, E_FAIL
    jz      shutdown
; Здесь мы должны создать окна, list view, цикл обработки сообщений и так
; далее...
; ....

; Очищение
; Освобождаем указатель на объект IMalloc
    mov     eax, g_pShellMalloc
    push    eax
    mov     eax, [eax]
    call    [eax + Release]         ; g_pShellMalloc->Release();

shutdown:
; закрываем библиотеку COM
    call    CoUninitialize

exit:
    push    wMsg.wParam
    call    ExitProcess
; Здесь программа прекращает свое выполнение
;-------------------------------------------------------
```

```
FillListView proc

; получаем папку Рабочего стола, сохраняем в pshf
    push    offset pshf
    call    SHGetDesktopFolder

; получаем объекты в папке Рабочего стола, используя метода EnumObjects
; объекта папки Рабочего стола
    push    offset peidl
    push    SHCONTF_NONFOLDERS
    push    0
    mov     eax, pshf
    push    eax
    mov     eax, [eax]
    call    [eax + EnumObjects]

    xor     ebx, ebx ; используем ebx в качестве счетчика

; перебираем элементы списка id
idlist_loop:
; Получаем следующий элемент списка
    push    0
    push    offset pidl
    push    1
    mov     eax, peidl
    push    eax
    mov     eax, [eax]
    call    [eax + Next]
    test    eax,eax
    jnz     idlist_endloop

    mov     lvi.imask, LVIF_TEXT or LVIF_IMAGE
    mov     lvi.iItem, ebx

; Получаем имя элемента, используя метод GetDisplayNameOf
    push    offset strret
    push    SHGDN_NORMAL
    push    offset pidl
    mov     eax, pshf
    push    eax
    mov     eax, [eax]
    call    [eax + GetDisplayNameOf]
; GetDisplayNameOf возвращает имя в одной из трех форм, поэтому выберите
; правильную форму и поступайте соответствующе
    cmp     strret.uType, STRRET_CSTR
    je      strret_cstr
    cmp     strret.uType, STRRET_OFFSET
    je      strret_offset

strret_olestr:
    ; здесь вы можете использовать WideCharToMultiByte, чтобы получить
    ; строку, я оставляю это на вас, так как я ленив
    jmp     strret_end

strret_cstr:
    lea     eax, strret.cStr
    jmp     strret_end

strret_offset:
    mov     eax, pidl
    add     eax, strret.uOffset

strret_end:
    mov     lvi.pszText, eax

; Получаем иконки элементов
    push    SHGFI_PIDL or SHGFI_SYSICONINDEX or SHGFI_SMALLICON or
SHGFI_ICON
    push    sizeof SHFILEINFO
    push    offset shfi
    push    0
```

```
    push    pidl
    call    SHGetFileInfo
    mov     eax, shfi.iIcon
    mov     lvi.iImage, eax

; теперь добавляем элементы в список
    push    offset lvi
    push    0
    push    LVM_INSERTITEM
    push    hWndListView
    call    SendMessage

; увеличиваем значение счетчика ebx и делаем еще один повтор цикла
    inc     ebx, ebx
    jmp     idlist_loop

idlist_endloop:

; теперь освобождаем список id
; Помните, что все зарезервированные объекты должны быть освобождены
    mov     eax, peidl
    push    eax
    mov     eax,[eax]
    call    [eax + Release]

; освобождаем объект папки Рабочего стола
    mov     eax, pshf
    push    eax
    mov     eax,[eax]
    call    [eax + Release]

    ret
FillListView endp

END start
```

Заключение

Хорошо, вот и все об использовании COM при программировании на ассемблере. Вероятно, что моя следующая статья расскажет о том, как определить собственные интерфейсы. Как вы можете видеть, использование COM вовсе не сложно, и с его помощью вы можете добавить очень мощные возможности в вашу программу, написанную на ассемблере.

COM в Ассемблере - Часть II

Автор: Bill T., пер. Aquila

Дата: 2002-06-27

В моей предыдущей статье объяснялось, как использовать COM-объекты в ваших программах, написанных на ассемблере. Там говорилось только о том, как вызывать COM-методы, а не как создавать свои COM-объекты. Эта статья расскажет, как это делать.

В этой статье будет затронуто реализация COM-объектов, используя синтаксис MASM. Здесь не будут браться в расчет ассемблеры TASM или NASM, хотя используемые методы легко применить к любому ассемблеру.

В этой статье не будут объсняться продвинутые технологии COM, такие как повтороное использование, трединг, серверы/клиенты и т.д. Об этом будет рассказано в будущих статьях.

Обзор интерфейсов COM

Определение интерфейса задает методы интерфейса, возвращаемые ими значения, количество и типы их параметров и что эти методы должны делать. Далее идет пример простого определения интерфейса:

```
IInterface struct
    lpVtbl  dd  ?
IInterface ends

IInterfaceVtbl struct
    ; методы IUnknown
    STDMETHOD       QueryInterface, :DWORD, :DWORD, :DWORD
    STDMETHOD       AddRef, :DWORD
    STDMETHOD       Release, :DWORD
    ; методы IInterface
    STDMETHOD       Method1, :DWORD
    STDMETHOD       Method2, :DWORD
IInterfaceVtbl ends
```

STDMETHOD используется для упрощения объявления интерфейса и определяется следующим образом:

```
STDMETHOD MACRO name, argl :VARARG
    LOCAL @tmp_a
    LOCAL @tmp_b
    @tmp_a TYPEDEF PROTO argl
    @tmp_b TYPEDEF PTR @tmp_a
    name @tmp_b ?
ENDM
```

Использование этого макроса значительно упрощает объявления интерфейсов и делает возможным использование команды invoke. (Макрос написан Ewald'ом :))

```
mov     eax, [lpif]                         ; lpif - указатель на интерфейс
mov     eax, [eax]                          ; получаем адрес vtable
invoke  (IInterfaceVtbl [eax]).Method1, [lpif] ; косвенный вызов функции
- or -
invoke  [eax][IInterfaceVtbl.Method2], [lpif]  ; альтернативная форма
                                               ; записи
```

Какую форму записи использовать - дело вкуса. В обоих случаях генерируется один и тот же код.

Все интерфейсы должны наследоваться от интерфейса IUnknown. Это означает, что первые 3 метода vtable должны быть QueryInterface, AddRef и Release. Цель и реализация этих методов будет обсуждаться ниже.

GUID'ы

GUID - это глобальный уникальный ID. GUID - это 16-ти байтное число, которое уникально у каждого интерфейса. COM использует GUID'ы, чтобы отличать интерфейсы друг от друга. Использование этого метода предотвращает проблемы с совпадением имен или версий. Чтобы получить GUID, вы можете использовать утилиту, которая включена в большинство пакетов разработки программ под Win32.

GUID можно представить как следующую структуру:

```
GUID STRUCT
    Data1   dd ?
    Data2   dw ?
    Data3   dw ?
    Data4   db 8 dup(?)
GUID ENDS
```

Затем GUID объявляется в секции данных:

```
MyGUID GUID <3F2504E0h, 4f89h, 11D3h, <9Ah, 0C3h, 0h, 0h, 0E8h, 2Ch, 3h,1h>>
```

Как только GUID ассоциирован с интерфейсом и опубликован, никаких дальнейших изменений в его определении быть не должно. Обратите внимание, это не означает, что не может меняться реализация интерфейса. Если необходимо изменение интерфейса, должен быть использован новый GUID.

COM-объекты

COM-объект - это просто реализация интерфейса. Детали реализации не затрагиваются стандартами COM, поэтому мы можем реализовывать объекты как угодно, пока это удовлетворяет всем требованиям определения интерфейса.

Типичный объект будет содержать указатели на различные интерфейсы, которые он поддерживает, счетчик ссылок и другие данные, которые могут потребоваться объекту. Вот простое определение объекта, реализованное в виде структуры:

```
Object struct
    interface   IInterface  <?>     ; указатель на IInterface
    nRefCount   dd          ?       ; счетчик ссылок
    nValue      dd          ?       ; приватные данные объекта
Object ends
```

Также мы должны определить vtable'ы, которые будут использоваться. Эти таблицы должны быть статическими и не могут меняться во время выполнения программы. Каждый член vtable - это указатель на метод. Далее показывается, как определить vtable.

```
@@IInterface segment dword
vtblIInterface:
    dd      offset IInterface@QueryInterface
    dd      offset IInterface@AddRef
    dd      offset IInterface@Release
    dd      offset IInterface@GetValue
    dd      offset IInterface@SetValue
@@IInterface ends
```

Подсчет ссылок

COM-объект управляет продолжительностью своей жизни с помощью подсчета ссылок. У каждого объекта есть счетчик ссылок, отслеживающий, как много экземпляров указателя на интерфейс было создано. Счетчик объекта должен поддерживать значение до 2^32, то есть он должен быть DWORD.

Когда счетчик ссылок падает до нуля, объект больше не используется и разрушает сам себя. Два метода интерфейса IUnknown AddRef и Release обрабатывают подсчет ссылок для COM-объекта.

QueryInterface

Метод QueryInterface используется, чтобы определить, поддерживается ли объектом определенный интерфейс, и если да, позволяет получить указатель на него. Есть три правила реализации метода QueryInterface:

1. Объекты должны быть идентичны - вызов QueryInterface должен всегда возвращать одно и то же значение.

2. Набор интерфейсов объекта не должен меняться - например, если вызов QueryInterface с неким IID был однажды успешен, то он должен быть успешным всегда. Таким же образом, если вызов однажды не удался, то не должен удасться в другой раз.

3. Должно быть возможно проверить наличие одного интерфейса объекта из другого интерфейса.

QueryInterface возвращает указатель на указанный интерфейс объекта, указатель на интерфейс которого уже есть у клиента. Эта функция должна вызывать метод AddRef указателя, который она возвращает.

Вот описание аргументов QueryInterface:

```
pif  : [in] указатель на вызывающий интерфейс
riid : [in] указатель на IID интерфейса, который запрашивается
ppv  : [out] указатель на указатель на интерфейс, который запрашивается.
       Если интерфейс не поддерживается, значение переменной будет
       приравнено 0.
```

QueryInterface возвращает следующее:

```
S_OK, если интерфейс поддерживается
E_NOINTERFACE, если не подерживается
```

Вот простая ассемблерная реализация QueryInterface:

```
IInterface@QueryInterface proc uses ebx pif:DWORD, riid:DWORD, ppv:DWORD
    ; Следующий код сравнивает затребованный IID с доступными. Так как
    ; интерфейс IInterface наследуется от IUnknown, эти два интерфейса
    ; разделяют один и тот же указатель на интерфейс.
    invoke  IsEqualGUID, [riid], addr IID_IInterface
    or      eax,eax
    jnz     @1
    invoke  IsEqualGUID, [riid], addr IID_IUnknown
    or      eax,eax
    jnz     @1
    jmp     @NoInterface

@1:
    ; GETOBJECTPOINTER - это макрос, который поместит указатель на объект
```

```
    ; в eax, если дано имя объекта, имя интерфейса и указатель на интерфейс.
    GETOBJECTPOINTER    Object, interface, pif

    ; теперь получаем указатель на затребованный интерфейс
    lea     eax, (Object ptr [eax]).interface

    ; заполняем *ppv указателем на интерфейс
    mov     ebx, [ppv]
    mov     dword ptr [ebx], eax

    ; повышаем значение счетчика ссылок, вызывая AddRef
    GETOBJECTPOINTER    Object, interface, pif
    mov     eax, (Object ptr [eax]).interface
    invoke  (IInterfaceVtbl ptr [eax]).AddRef, pif

    ; возвращаем S_OK
    mov     eax, S_OK
    jmp     return

@NoInterface:
    ; интерфейс не поддерживается, поэтому заполняем *ppv нулем
    mov     eax, [ppv]
    mov     dword ptr [eax], 0

    ; return E_NOINTERFACE
    mov     eax, E_NOINTERFACE

return:
    ret
IInterface@QueryInterface endp
```

AddRef

Метод AddRef используется для повышения значения счетчика ссылок для интерфейса объекта. Он должен вызываться для каждой новой копии указателя на интерфейс объекта.

AddRef не принимает параметров, кроме указателя на интерфейс, что требуется для всех методов. AddRef должен возвращать новое значение счетчика ссылок. Тем не менее, это значение должно использоваться вызывающими только в тестовых целях, так как в определенных ситуациях оно может быть нестабильно.

Далее идет простая реализация метода AddRef:

```
IInterface@AddRef proc pif:DWORD
    GETOBJECTPOINTER    Object, interface, pif
    ; увеличиваем значение счетчика ссылок
    inc     [(Object ptr [eax]).nRefCount]
    ; теперь вовращяем значение ссылок
    mov     eax, [(Object ptr [eax]).nRefCount]
    ret
IInterface@AddRef endp
```

Release

Release понижает значение счетчика ссылок вызывающего интерфейса объекта. Если значение счетчика объекта снижается до 0, то объект выгружается из памяти. Эта функция должна вызываться, когда в указателе на интерфейс больше нет надобности.

Как и AddRef, Release принимает только один аргумент - указатель на интерфейс. Он также возвращает текущее значение счетчика ссылок, который также следует использовать только для

тестирования.

Вот простая реализация Release:

```
IInterface@Release proc pif:DWORD
    GETOBJECTPOINTER    Object, interface, pif

    ; decrement the reference count
    ; понижаем значение счетчика ссылок
    dec     [(Object ptr [eax]).nRefCount]

    ; проверяем, равно ли значение счетчика ссылок нулю. Если так, то
    ; выгружаем объект
    mov     eax, [(Object ptr [eax]).nRefCount]
    or      eax, eax
    jnz     @1

    ; освобождаем объект: здесь мы предполагаем, что объект был
    ; зарезервирован функцией LocalAlloc и с опцией LMEM_FIXED
    GETOBJECTPOINTER    Object, interface, pif
    invoke  LocalFree, eax
@1:
    ret
IInterface@Release endp
```

Создание COM-объекта

Создание объекта состоит из резервирования памяти для объекта, а затем инициализации его данных. Как правило, инициализируется указатель на vtable и обнуляется счетчик ссылок. Затем можно вызывать QueryInterface, чтобы получить указатель на интерфейс.

Есть и другие методы для создания объектов, такие как CoCreateInstance и использование фабрик классов. Эти методы не будут обсуждаться в данной статье.

Пример реализации COM-объекта

Здесь приводится простая реализация COM-объекта. Демонстрируется, как создать объект, вызвать его методы, а затем освободить их. Вероятно, будет довольно познавательно скомпилировать данный пример и пройтись по нему отладчиком.

```
.386
.model flat,stdcall

include windows.inc
include kernel32.inc
include user32.inc

includelib kernel32.lib
includelib user32.lib
includelib uuid.lib

;-------------------------------------------------------------------------

; Borrowed from Ewald, http://here.is/diamond/
; Макрос для упрощения объявлений интерфейсов
STDMETHOD   MACRO   name, argl :VARARG
LOCAL @tmp_a
LOCAL @tmp_b
@tmp_a  TYPEDEF PROTO argl
@tmp_b  TYPEDEF PTR @tmp_a
name    @tmp_b      ?
```

```
ENDM

; Макрос, который получает указатель на интерфейс и возвращает указатель
; на реализацию в eax
GETOBJECTPOINTER MACRO Object, Interface, pif
    mov     eax, pif
    IF (Object.Interface)
        sub     eax, Object.Interface
    ENDIF
ENDM

;-------------------------------------------------------------------------

IInterface@QueryInterface   proto :DWORD, :DWORD, :DWORD
IInterface@AddRef           proto :DWORD
IInterface@Release          proto :DWORD
IInterface@Get              proto :DWORD
IInterface@Set              proto :DWORD, :DWORD

CreateObject                proto :DWORD
IsEqualGUID                 proto :DWORD, :DWORD

externdef                   IID_IUnknown:GUID

;-------------------------------------------------------------------------

; объявляем прототип интерфейса
IInterface struct
    lpVtbl  dd  ?
IInterface ends

IInterfaceVtbl struct
    ; методы IUnknown
    STDMETHOD       QueryInterface, pif:DWORD, riid:DWORD, ppv:DWORD
    STDMETHOD       AddRef, pif:DWORD
    STDMETHOD       Release, pif:DWORD
    ; методы IInterface
    STDMETHOD       GetValue, pif:DWORD
    STDMETHOD       SetValue, pif:DWORD, val:DWORD
IInterfaceVtbl ends

; объявляем структуру объекта
Object struct
    ; интерфейс объекта
    interface   IInterface  <?>

    ; данные объекта
    nRefCount   dd          ?
    nValue      dd          ?
Object ends

;-------------------------------------------------------------------------

.data
; define the vtable
; определяем vtable
@@IInterface segment dword
vtblIInterface:
    dd      offset IInterface@QueryInterface
    dd      offset IInterface@AddRef
    dd      offset IInterface@Release
    dd      offset IInterface@GetValue
    dd      offset IInterface@SetValue
@@IInterface ends

; определяем IID интерфейса
; {CF2504E0-4F89-11d3-9AC3-0000E82C0301}
IID_IInterface GUID <0cf2504e0h, 04f89h, 011d3h, <09ah, 0c3h, 00h, 00h, \
                     0e8h, 02ch, 03h, 01h>>

;-------------------------------------------------------------------------
```

```
.code
start:
StartProc proc
    LOCAL   pif:DWORD       ; указатель на интерфейс

    ; вызываем метод SetValue
    mov     eax, [pif]
    mov     eax, [eax]
    invoke  (IInterfaceVtbl ptr [eax]).SetValue, [pif], 12345h

    ; вызываем метод GetValue
    mov     eax, [pif]
    mov     eax, [eax]
    invoke  (IInterfaceVtbl ptr [eax]).GetValue, [pif]

    ; освобождаем объект
    mov     eax, [pif]
    mov     eax, [eax]
    invoke  (IInterfaceVtbl ptr [eax]).Release, [pif]

    ret
StartProc endp

;------------------------------------------------------------------------------

IInterface@QueryInterface proc uses ebx pif:DWORD, riid:DWORD, ppv:DWORD
    invoke  IsEqualGUID, [riid], addr IID_IInterface
    test    eax,eax
    jnz     @F
    invoke  IsEqualGUID, [riid], addr IID_IUnknown
    test    eax,eax
    jnz     @F
    jmp     @Error

@@:
    GETOBJECTPOINTER    Object, interface, pif
    lea     eax, (Object ptr [eax]).interface

    ; устанавливаем значение *ppv
    mov     ebx, [ppv]
    mov     dword ptr [ebx], eax

    ; увеличиваем значение счетчика ссылок
    GETOBJECTPOINTER    Object, interface, pif
    mov     eax, (Object ptr [eax]).interface
    invoke  (IInterfaceVtbl ptr [eax]).AddRef, [pif]

    ; возвращаем S_OK
    mov     eax, S_OK
    jmp     return

@Error:
    ; ошибка, интерфейс не поддерживается
    mov     eax, [ppv]
    mov     dword ptr [eax], 0
    mov     eax, E_NOINTERFACE

return:
    ret
IInterface@QueryInterface endp


IInterface@AddRef proc pif:DWORD
    GETOBJECTPOINTER    Object, interface, pif
    inc     [(Object ptr [eax]).nRefCount]
    mov     eax, [(Object ptr [eax]).nRefCount]
    ret
IInterface@AddRef endp


IInterface@Release proc pif:DWORD
```

```
    GETOBJECTPOINTER    Object, interface, pif
    dec     [(Object ptr [eax]).nRefCount]
    mov     eax, [(Object ptr [eax]).nRefCount]
    or      eax, eax
    jnz     @1
    ; free object
    mov     eax, [pif]
    mov     eax, [eax]
    invoke  LocalFree, eax
@1:
    ret
IInterface@Release endp


IInterface@GetValue proc pif:DWORD
    GETOBJECTPOINTER    Object, interface, pif
    mov     eax, (Object ptr [eax]).nValue
    ret
IInterface@GetValue endp


IInterface@SetValue proc uses ebx pif:DWORD, val:DWORD
    GETOBJECTPOINTER    Object, interface, pif
    mov     ebx, eax
    mov     eax, [val]
    mov     (Object ptr [ebx]).nValue, eax
    ret
IInterface@SetValue endp

;------------------------------------------------------------------------

CreateObject proc uses ebx ecx pobj:DWORD
    ; set *ppv to 0
    mov     eax, pobj
    mov     dword ptr [eax], 0

    ; резервируем объект
    invoke  LocalAlloc, LMEM_FIXED, sizeof Object
    or      eax, eax
    jnz     @1
    ; alloc failed, so return
    mov     eax, E_OUTOFMEMORY
    jmp     return
@1:

    mov     ebx, eax
    mov     (Object ptr [ebx]).interface.lpVtbl, offset vtblIInterface
    mov     (Object ptr [ebx]).nRefCount, 0
    mov     (Object ptr [ebx]).nValue, 0

    ; Запрашиваем интерфейс
    lea     ecx, (Object ptr [ebx]).interface
    mov     eax, (Object ptr [ebx]).interface.lpVtbl
    invoke  (IInterfaceVtbl ptr [eax]).QueryInterface,
            ecx,
            addr IID_IInterface,
            [pobj]
    cmp     eax, S_OK
    je      return

    ; ошибка в QueryInterface, поэтому освобождаем память
    push    eax
    invoke  LocalFree, ebx
    pop     eax

return:
    ret
CreateObject endp

;------------------------------------------------------------------------
IsEqualGUID proc rguid1:DWORD, rguid2:DWORD
```

```
    cld
    mov     esi, [rguid1]
    mov     edi, [rguid2]
    mov     ecx, sizeof GUID / 4
    repe    cmpsd
    xor     eax, eax
    or      ecx, ecx
    setz    eax
    ret
IsEqualGUID endp

end start
```

Заключение

Мы увидели (надеюсь), как реализовать COM-объект. Мы можем видеть, что это связанно с определенными трудностями и увеличивает избыточность кода нашей программы. Тем не менее, это может добавить гибкость и силу в наши программы. Подробности по данной теме вы можете найти на моей маленькой странице, посвященной COM: http://lordlucifer.cjb.net.

Помните, что COM определяет только интерфейсы, а реализацию оставляет на программиста. Эта статья показывает только одну возможную реализацию. Это не единственный метод и не самый лучший. Читатель не должен бояться экспериментировать с другими методами.

Как получить доступ к COM-объекту, используя ассемблер

Автор: Ernie, пер. Aquila

Дата: 2002-06-27

Предисловие:

COM (Component Object Model) используется операционной системой Windows различным образом. Например, shell.dll использует COM для доступа к некоторым из своих API методов. Интерфейсы IShellLink и IPersistFile shell32.dll будут продемонстрированы путем создания ярлыка. Предполагается базовое знание COM. Использованный код выдержан в стиле MASM.

Введение:

COM может показаться весьма сложным из-за своих многочисленных особенностей, но в реальности многие из этих сложностей вырождаются в простой вызов функций. Самой сложной частью является понимание вовлеченных в процесс структур данных, которые необходимо использовать при определении интерфейсов. Я извиняюсь за всю эту C++'нутую терминологию, использованную в этом туториале. Хотя COM является языконезависимой технологией, большая часть терминологии для его определения была заимствована из C++.

Чтобы использовать COM-методы какого-либо объекта, сначала вы должны инстанцировать или создать этот объект из соответствующего com-класса, затем получить от него указатель на его интерфейс. Этот процесс производится с помощью API-функции CoCreateInstance. После того, как вы выполните все, что нужно, вам необходимо вызвать его метод Release, и com-класс удалит объект, а COM выгрузит com-класс.

COM-объект называется сервером, а программа, которая его вызывает, называется клиентом.

Доступ к COM-методам

Чтобы использовать COM-методы, вам необходимо знать структуру интерфейса. Даже если вы используете "позднее связывание" через IDispatch, вам все равно нужно знать структуру IDispatch. COM-интерфейс - это всего лишь таблица указателей на функции. Давайте начнем с интерфейса IUnknown. Если бы вы создавали компонент, который просто экспортирует интерфейс IUnknown, то получили бы полностью рабочий COM-объект (хотя и на уровне "Hello, world"). IUnknown имеет три основных метода, которые есть в любом интерфейсе, потому что любой интерфейс наследуется от IUnknown. Держите в уме, что все интерфейсы являются структурой указателей на функции. Для IUnknown это будет выглядеть так:

```
IUnknown STRUCT DWORD
; IUnknown methods
IUnknown_QueryInterface         QueryInterface_Pointer ?
IUnknown_AddRef                 AddRef_Pointer ?
IUnknown_Release                Release_Pointer ?
IUnknown ENDS
```

Именно так, всего лишь 12 байтов длиной. Он состоит из трех 4-х байтных указателей на процедуры, которые фактически и являются методами. Это и есть печально известная "vtable", о которой вы уже могли слышать. Указатели определены так, чтобы MASM мог делать для нас кое-какую проверку типов при компиляции наших вызовов. Так как виртуальная таблица содержит адреса функций (указатели), эти указатели определены в нашем описании интерфейса через typedef.

```
QueryInterface_Pointer  typedef ptr QueryInterface_Proto
AddRef_Pointer          typedef ptr AddRef_Proto
Release_Pointer                 typedef ptr Release_Proto
```

Наконец мы определяем прототипы функций следующим образом:

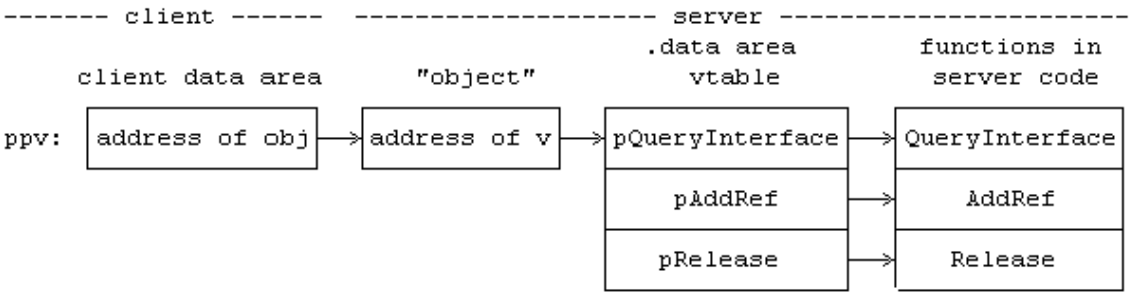
```
QueryInterface_Proto    typedef PROTO :DWORD, :DWORD, :DWORD
AddRef_Proto            typedef PROTO :DWORD
Release_Proto                   typedef PROTO :DWORD
```

Подобное определение интерфейса позволяет выявлять множество ошибок в процессе компиляции. На практике мы определять эти интерфейсы в заголовочных файлах, чтобы они не засоряли исходный код.

Одно довольно большое замечание относительно определения интерфейсов: MASM не может использовать подобные косвенные ссылки, поэтому мы определяем заранее прототипы функций через typedef'ы, а потом виртуальную таблицу интерфейса. В заголовочных файлах программы-примера интерфейсы определяются именно таким образом.

Чтобы использовать интерфейс, вам нужно получить на него указатель.

Функция CoCreateInstance может быть использована для получения этого косвенного указателя на структуру интерфейса. Таким образом, один уровень "виртуальности" убран и у нас есть указатель на "объект", который содержит структуру. Окончательно все это выглядит так:



Как видите, здесь очень много косвенных ссылок, что может заставить вас спятить, пока вы будете писать код, в котором все элементы будут ссылаться друг на друга правильно, поэтому в дальнейшем будут определены макросы для облегчения данной задачи.

Когда клиент делает вызов на COM-библиотеку, чтобы создать COM-объект, он передает адрес переменной, в которую будет помещен указатель на объект. Этот изначальный указатель часто называют "ppv", от C++'нутого "pointer to pointer to (void)", где (void) значит неопределенный тип. Ppv содержит адрес другого указателя ("pv"), который, в свою очередь ссылается на всю таблицу указателей. В ней на каждую функцию интерфейса приходится один элемент таблицы.

Например, предположим, что мы использует CoCreateInstance и успешно получили интерфейсный указатель ppv и узнать, поддерживает ли он другие интерфейсы. Мы можем вызвать его метод QueryInterface и запросить новый ppv (ppv2, указатель на интерфейс) на другой интерфейс (pIID, указатель на Interface Indentifying GUID), который нам нужен. В C++ QueryInterface имеет следующий прототип:

(HRESULT) SomeObject::QueryInterface (this:pObject, IID:pGUID, ppv2:pInterface)

Подобный вызов будет выглядеть примерно так:

```
; получаем указатель на объект
mov eax, ppv
; и используем его, чтобы найти структуру интерфейса
mov edx, [eax]

; загоняем параметры функции в стек
push OFFSET ppv2
push OFFSET IID_ISomeOtherInterface
push dword ppv

; а затем вызываем этот метод
call dword ptr [eax + 0]
```

Это может быть выполнено с помощью встроенного в MASM макроса 'invoke' следующим образом:

```
; получаем указатель на объект
mov eax, ppv
; а затем используем его, чтобы найти структуру интерфейса
mov edx, [eax]
; а затем вызываем этот метод
invoke (IUnknown PTR [edx]).IUnknown_QueryInterface, ppv,
    ADDR IID_SomeOtherInterface, ADDR ppv_new
```

Я надеюсь, что это для будет настолько же просто, как и для меня.

Обратите внимание, что мы должны передать функции использованный нами указатель, так как это позволяет знать интерфейсу какой объект (в C++ этот указатель обозначается как "this") мы используем.

Также заметьте, что необходимо делать привод регистра к соответсвующему типу. Это позоволяет компилятору знать, какую структуру использовать, чтобы получить правильное смещение в виртуальной таблице для функции .QueryInterface (в данном случае это означает нулевое смещение от [edx].

Еще один момент, который может быть не совсем ясен. Обратите внимание, что я изменил имя функции с "QueryInterface" на "IUnknown_QueryInterface". Я считаю подобное изменение имени функций необходимым, так как когда вы перейдете к COM-проектам со множеством похожих интерфейсов, в которых будут встречаться методы с одинаковыми именами. В этом нет ничего неправильного, фактически это и есть то, что называется полиморфизмом, но это может немного смутить компилятор.

Макрос coinvoke

```
;--------------------------------------------------------------------------
С помощью этого макроса мы можем упростить вызов COM-методов. Этот макрос
является частью файла oaidl.inc.
;--------------------------------------------------------------------------
; coinvoke MACRO
;
; invokes an arbitrary COM interface
; Запускает метод из заданного COM-интерфейса
;
; revised 12/29/00 to check for edx as a param and force compilation error
;                  (thanks to Andy Car for a how-to suggestion)
; revised 7/18/00 to pass pointer in edx (not eax) to avoid confusion with
;   parmas passed with ADDR  (Jeremy Collake's excellent suggestion)
; revised 5/4/00 for member function name decoration
; see http://ourworld.compuserve.com/homepages/ernies_world/coinvoke.htm
;
; pInterface    pointer to a specific interface instance
; pInterface
; Interface     the Interface's struct typedef
; Function      which function or method of the interface to perform
; args          all required arguments
;                   (type, kind and count determined by the function)
;
coinvoke MACRO pInterface:REQ, Interface:REQ, Function:REQ, args:VARARG
    LOCAL istatement, arg
    FOR arg, <args>     ;; run thru args to see if edx is lurking in there
        IFIDNI <&arg>, <edx>
            .ERR <edx is not allowed as a coinvoke parameter>
        ENDIF
    ENDM
    istatement CATSTR <invoke (Interface PTR[edx]).&Interface>,<_>,<&Function, pInterface>
    IFNB <args>     ;; add the list of parameter arguments if any
        istatement CATSTR istatement, <, >, <&args>
    ENDIF
    mov edx, pInterface
    mov edx, [edx]
    istatement
ENDM
;--------------------------------------------------------------------------
```

Таким образом, тот же метод QueryInterface теперь можно вызвать одной строчкой кода:

```
coinvoke ppv ,IUnknown, QueryInterface, ADDR IID_SomeOtherInterface, ADDR ppnew
```

Обратите внимание, что теперь преобразованием имени делает макрос.

Единственный 'подводный камень' (хорошо, наиболее очевидный) - это то, что параметры COM-функции не должны передаваться через edx, так как он используется для хранения ссылки на 'this'. Использование edx'a в качестве параметра сгенерирует ошибку компиляции.

Использование IShellFile и IPersistFile из shell32.dll

Shell32.dll предоставляет простой, легкий путь для создания ярлыков. Тем не менее, она использует COM-интерфейс для этого. Приведенный ниже пример основан на секции MSDN "Shell Links" из "Internet Tools and Technologies".

Эту статью вы можете найти по адресу http://msdn.microsoft.com/library/psdk/shellcc/shell/Shortcut.htm.

Мы получим доступ к нескольким членам интерфейсов IShellLink и IPersistFile. Обратите внимание, что каждый интерфейс включает параметр "ppi", это интерфейс, который мы вызываем (это

параметр THIS). (Следующая информация является копией той, что предоставленна Микрософтом).

• IShellLink::QueryInterface, ppi, ADDR riid, ADDR ppv
• riid: идентификатор требуемого интерфейса.
• ppv: указатель на переменную, которая получает интерфейс.
• Описание: проверяет, поддерживает ли объект требуемый интерфейс. Если это так, то заполняет ppv указателем на интерфейс.
• IShellLink::Release, ppi
• Описание: понижает количество ссылкок на интерфейс IShellLink на один.
• IShellLink::SetPath, ppi, ADDR szFile
• pszFile: указатель на текстовый буфер, содержащий новый путь на объект-ярлык.
• Описание: задает путь к файлу, на который указывает ярлык.
• IShellLink::SetIconLocation, ppi, ADDR szIconPath, iIcon
• pszIconPath: указатель на текстовый буфер, содержащий новый путь к иконке.
• iIcon: индекс иконки. Отсчет идет с нуля.
• Description: устанавливает какую иконку будет использовать ярлык.
• IPersistFile::Save, ppi, ADDR szFileName, fRemember
• pszFileName: указывает на ASCIIZ-строку, содержащую абсолютный путь к файлу, в который должен быть сохранен объект.
• fRemember: указывает, должен ли файл, заданный в pszFileName, стать текущим рабочим файлом. Если TRUE, то pszFileName становится текущим файлом и объект должен очистить свой флаг занятости после сохранения. Если FALSE, то это сохраняет операцию как "Save A Copy As ...". В этом случае текущий файл не изменяется, а объект не должен очищаеть свой флаг занятости. Если pszFileName равен NULL, флаг fRemember должен игнорироваться.
• Описание: выполняет операцию сохранения для объекта-ярлыка или сохраняет созданный ярлык.
• IPersistFile::Release, ppi
• Description: уменьшает счетчик ссылок на интерфейс IPersistFile на один. Эти интерфейсы содержат гораздо больше методов (смотри полное определение интерфейса в прилагающемся к туториалу коде), но мы сконцентрируемся только на тех, которые реально используем.

В файле ярлыка (.lnk) содержится следующая информацию:

• Путь к файлу и имя программы
• Откуда берется иконка, используемая при отображении ярлыка (обычно берется из самого экзешника), и какая конкретна иконка берется из этого файла. Мы будем использовать первую иконку из этого файла.
• Путь к файлу, в котором должен быть сохранен ярлык.

Использование этих интерфейсов достаточно просто и прямолинейно. Это выглядит примерно следующим образом:

• Вызываем CoCreateInstance CLSID_ShellLink, чтобы получить интерфейс IID_IShellLink, Queryinterface IShellLink, чтобы получить интерфейс IID_IPersistFile.
• Вызываем IShellLik.SetPath, чтобы указать, где находится файл, на который будет ссылаться ярлык.
• Вызываем IShellLink.SetIconLocation, чтобы указать, какую иконку использовать.
• Вызываем IPersistFile.Save, чтобы сохранить наш ярлык.
• Вызываем IPersistFile.Release.
• Вызываем IShellLink.Release.

Вам следуется знать о нескольких полезных утилитах, входящий в MSVC: консольная утилита "FindGUID.exe", которая просматривает регистр в поисках определенного интерфейса или com-класса или выведет список всех классов и интерфейсов с ассоциированными с ними GUID'ами,

а также приложение OLEView, которамя позволит вам получать информацию из различных библиотек.

Будьте внимательны во время определения интерфейса. Пропуск методы в виртуальной таблице может повлечь странные результаты. Выкиньте метод из определения таблице вы получите неверный интерфейс. В оригинальное определении интерфейса IShellLink, которое я использовал, была пропущена одна функция. Вызовы, которые я осуществлял, возвращали в hResult "SUCCEDED", но в некоторых случаях стек очищался неправильно (поскольку не совпадало количество параметров, а это, в свою очередь, приводил к GPF, как только я выходил из процедуры. Держите это в уме, если у вас начинают происходить "странные" вещи.

MakeLink.asm, демонстрация COM

Эта программа делает очень немного, как и полагается хорошей программе-туториалу. Как только вы ее запустите, она создаст ярлык на саму себя в той же директории. Довольно необычно запускать ее из файлового менеджера и смотреть, как появляется ярлык. Затем вы можете испытать ярлык и увидеть, как изменится время его создания.

Исходник программы находится в masm32\COM\examples\shortcut. Он начинается с "хакерского" кода, с помощью которого программа получает полный путь к программе и формирует необходимые строки, которые затем передаются интерфейсу IShellLink, после чего ярлык сохраняется.

Процедура CoCreateLink используется для запуска соответствующих COM-методов и создания ярлыка так, чтобы эта процедура могла использоваться в других программах.

Введение в использование скриптовых движков: Часть 1

Автор: Hangatyr

Дата: 2004-01-03

Предисловие В этой статье (а возможно и цикле статей) я попытаюсь рассказать о том, как добавить поддержку скриптовых языков в программу, написанную на ассемблере. Наверное, это самый простой путь реализовать поддержку макросов в своей программе и, может быть, кому-то это и пригодится ;). **Пара слов о компиляторе**

Я не буду здесь поднимать бесполезный вопрос о том какой компилятор самый лучше и т.п. - в конце концов это дело вкуса. Скажу только, что я буду использовать fasm (http://flatassembler.net) и, соответственно, полагаю, что для читателя не будет сюрпризом увидеть

```
add eax, dword [ecx]
```

вместо

```
add eax, dword ptr [ecx]
```

Но вообще-то это, наверное, не проблема ;)

Colib Говоря о использовании COM в ассемблере нельзя не сказать о работах Ernest Murphy, в том числе о библиотеке Colib, которая предоставляет нам набор удобных функций для работы с COM-объектами. Но у нее есть один минус - она ориентирована на MASM. Хотя минус это или плюс решать не мне и если ваш любимый ассемблер - MASM32 (с которым, кстати, и поставляется вышеозначенная библиотека), то вы вполне можете радоваться, т.к. часть работы уже сделана за вас. Но изучить прилагающиеся к ней примеры не помешает в любом случае. Еще советую почитать статью "Using COM in Assembly Language" by Bill Tyler. **Основы Component Object Model** В этом разделе я постараюсь кратко объяснить некоторые основы технологии COM, которые необходимы для понимания материала, изложенного в данной статье. Как обычно, те кто уже имеет нужные знания (а нужно нам не так уж и много) могут спокойно пропустить этот раздел и перейти сразу ко второй части. А для оставшихся продолжаем. *Введение*

Технология COM - это мощный инструмент, позволяющий разбить монолитное приложения на самостоятельные компоненты. Такая необходимость неизбежно должна была возникнуть, учитывая темпы развития индустрии программирования.

Что же такое COM? Не более, чем набор правил, определяющих каким образом могут взаимодействовать компоненты, написанные в общем случае разными людьми и на разных языках программирования, это стандарт, призванный обеспечить возможность их совместной работы. Идея, лежащая в основе, довольно проста - код, который должен стать всеобщим достоянием оформляется в виде своеобразного мини-приложения, так называемого COM-объекта. А функции, которые должны быть доступны извне группируются в интерфейсы этого объекта. Через них и осуществляется взаимодействие с клиентами. Интерфейсов может быть сколько угодно - правило здесь только одно - спецификация COM требует, чтобы каждый объект содержал специальный интерфейс IUnknown, а также чтобы любой другой интерфейс так или иначе наследовался от него.

Интерфейс IUnknown

Этот интерфейс занимает в COM особое место и это не случайно - с его помощью можно всегда получить остальные интерфейсы, поддерживающиеся объектом, кроме этого он предоставляет нам некоторые базовые средства управления жизнью объекта. Вот 3 его метода (функции интерфейсов называются методами):

_ **HRESULT QueryInterface(REFIID iid, void** ppvObject) - данный метод используется для получение указателя на интерфейс объекта по идентификатору. Если запрошенный интерфейс поддерживается объектом, то будет возвращено значение S_OK и указатель на интерфейс помещен в ppvObject. В противном случае возвращается E_NOINTERFACE._

_ **ULONG AddRef(void)** - инкрементирует счетчик использования объекта и должен вызываться всякий раз, когда некто собирается использовать объект (например, он должен вызывается всякий раз, когда метод QueryInterface возвращает указаетель на какой-нибудь интерфейс объекта). Возвращаемое значение - новое значение счетчика ссылок._

_ **ULONG Release(void)** - этот метод выполняет обратные действия и должен вызываться, когда клиент закончил работу с объектом._

Как было сказано выше, эти методы должны быть первыми методами любого интерфейса и именно в таком порядке (порядок следования очень важен - это следует из структуры интерфейса, принятой в COM).

Структура объектов и интерфейсов

Так как спецификация COM не предусматривает никаких особых стандартов в отношении реализации объектов, то мы не будем заострять на этом внимания, скажу лишь, что любой объект можно условно разделить на две части - первая содержит указатели на интерфесы, а вторая - различные данные (в том числе и вышеупомянутый счетчик ссылок).

А вот интерфейсы в отличии от объектов должны иметь четко оговоренную структуру.

Основой интерфейса является массив адресов функций (часто называемый просто "vtable"). В качестве примера рассмотрим простейший интерфейс, состоящий из четырех методов (три из них - методы IUnknown).

```
...
proc Addref, pi
enter
...
return
...

proc Method1, pi
enter
...
return
...

IsomeInterface  dd  IsomeInterface_vtable

IsomeInterface_vtable dd  QueryInterface
    dd  Addref
    dd  Release
    dd  Method1
```

Довольно просто, не так ли? Чтобы вызвать определенный метод нужно знать только его положение в vtable, правда тут есть одна тонкость - первым параметром методу всегда передается указатель на интерфейс - даже если в описании метода значится void т.е. вызов Release будет выглядеть не так:

```
mov eax, [IsomeInterface]
   call dword [eax + 8]
```

а вот так:

```
    mov eax, [IsomeInterface]
    push Isome_interface
    call dword [eax + 8]
```

Дело в том, что описания методов даются, как правило, для языков высокого уровня, а там эту работу за нас выполняет компилятор и, например, на C аналогичный вызов действительно будет таким:

```
    pIUnknown->Release();
```

Script Engines.

Предположим, что у нас в наличии есть такая программка:

(Легко заметить, что она ничего не делает кроме того, что показывает окно открытия файла по нажатию на кнопку)

```
format PE GUI 4.0
entry start
include 'include\win32ax.inc'
section '.scrpt' code readable executable

start:      invoke GetModuleHandleA, 0
  mov [hInst], eax
  invoke DialogBoxParamA, eax, 1000, HWND_DESKTOP, DlgProc, 0
  invoke ExitProcess, 0

proc DlgProc, hwnd, msg, wparam, lparam
  enter
  cmp [msg], WM_INITDIALOG
  jz wminitdialog
  cmp [msg], WM_COMMAND
  jz wmcommand
  cmp [msg], WM_CLOSE
  jz wmclose
  sub eax, eax
  jmp finish

wminitdialog:
mov eax, [hwnd]
  mov [hWnd], eax
  invoke VirtualAlloc, 0, MAX_PATH, MEM_COMMIT, PAGE_READWRITE
  mov [szFileName], eax

  jmp processed

wmcommand: cmp [wparam], BN_CLICKED shl 16 + 1001
  jnz processed

  mov eax, [szFileName]
  mov ecx, MAX_PATH
@@:  mov byte [eax], 0
  inc eax
  dec ecx
  jnz @B

  push szFilter
  push szTitle
  push dword [hwnd]
  call GetFileName

  jmp processed

wmclose: invoke VirtualFree, [szFileName], MAX_PATH, MEM_DECOMMIT
            invoke EndDialog, [hwnd], 0

processed: sub eax, eax
  inc eax

finish: return

proc GetFileName, hParent, lpTitle, lpFilter

  enter
```

```
  mov eax, [hParent]
      mov [ofn.hwndOwner], eax
     mov eax, [hInst]
     mov [ofn.hInstance], eax
     mov eax, [lpFilter]
      mov [ofn.lpstrFilter], eax
      mov eax, [szFileName]
      mov [ofn.lpstrFile], eax
      mov [ofn.nMaxFile], MAX_PATH
mov eax, [lpTitle]
      mov [ofn.lpstrTitle], eax
      mov [ofn.Flags], OFN_EXPLORER or OFN_FILEMUSTEXIST or OFN_LONGNAMES

  invoke GetOpenFileName, ofn
  return

section '.data' data readable writeable

ofn OPENFILENAMEA

szFilter db 'VBScripts', 0, '*.vbs', 0, \
     'All files (heh... you can try...)', 0, '*.*', 0, 0

szTitle db 'Just open file...', 0
szFileName dd 0
hInst dd 0
hWnd dd 0
section '.idata' import data readable writeable
library kernel32, 'kernel32.dll', \
 user32, 'user32.dll', \
       comdlg32, 'comdlg32.dll'

import kernel32, ExitProcess, 'ExitProcess', \
   VirtualAlloc, 'VirtualAlloc', \
          VirtualFree, 'VirtualFree', \
  GetModuleHandleA, 'GetModuleHandleA'

import user32, DialogBoxParamA, 'DialogBoxParamA', \
        EndDialog, 'EndDialog', \
        MessageBoxA, 'MessageBoxA'

import comdlg32, GetOpenFileName, 'GetOpenFileNameA'

section '.rsrc' resource data readable

directory RT_DIALOG, dialogs

resource dialogs, 1000, LANG_ENGLISH+SUBLANG_DEFAULT, main_dlg

dialog main_dlg, 'RunScript', 70, 70, 70, 50, WS_CAPTION+WS_POPUP+WS_SYSMENU+DS_MODALFRAME
  dialogitem 'BUTTON', 'Execute Script!', 1001, 8, 15, 55, 15, WS_VISIBLE+WS_TABSTOP+BS_PUSHBUTTON
enddialog
```

А также имеется скрипт:

```
Dim a, b
a = InputBox("A:")
b = MsgBox("A^2=" & a^2, 64, "--")
```

(Да простят меня Боги за появление здесь этих крамольных строк ;)

И кроме этого есть еще желание заставить его выполнятся. Ну что же приступим...

На сегодняшний день всеми нами любимая корпорация Microsoft предоставляет скиптовые движки JavaScript и VBScript совершенно свободно - единственное, чего они хотят, так это упоминание о них в AboutBox'e программы, что, согласитесь, довольно нетипично.

Физически эти движки находятся в файлах jscript.dll и vbscript.dll, но оформлены в виде COM-объектов, а значит наша цель - написать приложение, которое может работать с COM, чем мы сейчас и займемся.

Самое первое, что нужно сделать - это поставить вызов функции **CoInitialize** где-нибудь в начале, например, при создании окна:

```
wminitdialog: mov eax, [hwnd]
   mov [hWnd], eax
   invoke CoInitialize, 0
```

Данная функция инициализирует библиотеку COM и должна быть вызвана перед использованием любых CoXXX-функций.

Теперь с помощью API **CoCreateInstance** попросим систему создать для нас объект, который предоставит интерфейсы для управления движком VBScript.

```
invoke CoCreateInstance, CLSID_VBScript, 0, CLSCTX_INPROC_SERVER, \
       IID_IActiveScript, pIActiveScript
```

Первый параметр представляет собой идентификатор класса, к которому принадлежит создаваемый объект. Нужный нам объект принадлежит к классу "VB Script Language", имеющему ID **0b54f3741h-5b07h-11cfh-0a4b0h-00aa004a55e8h** Второй параметр указывает где должен быть запущен код, создающий объект. Значение **CLSCTX_INPROC_SERVER** говорит, чтобы это было сделано в контексте нашего процесса.

Третий представляет собой идентификатор запрашиваемого интерфейса. В нашем случае он должен быть таким: **0bb1a2ae1h-0a4f9h-11cfh-8fh20h-0805f2c0d064** Как вы уже знаете, COM-объекты доступны только через свои интерфейсы и нужный нам VBScript здесь не исключение - для управления движком нам предоставляются два интерфейса (хотя, если быть точным, то их больше, но здесь мы рассмотрим только два): **IActiveScript** - содержит методы для инициализации движка и управления его состоянием. А с помощью **IActiveScriptParse** мы можем указать скрипт, который нужно выполнить.

В случае успешного заверешения вызова **CoCreateInstance** в pIActiveScript будет помещен указатель на интерфейс **IActiveScript**, который мы запросили, указав GUID **0bb1a2ae1h-0a4f9h-11cfh-8fh20h-0805f2c0d064**.

Но нам нужен еще и интерфейс **IActiveScriptParse**, давайте его сейчас получим. Так как он является интерфесом того же объекта, что и **IActiveScript** мы можем воспользоваться методом QueryInterface последнего.

```
mov edx, [pIActiveScript]
mov edx, [edx]
push pIActiveScriptParse
push IID_IActiveScriptParse
push [pIActiveScript]
call dword [edx]
```

Этот код поместит указатель на **IActiveScriptParse** в переменную pIActiveScriptParse.

Теперь у нас есть все, чтобы управлять скриптовым движком. Думаете это все? Самое интересное еще впереди ;)

IActiveScriptSite и IActiveScriptSiteWindow.

Теперь у нас есть движок, исполняющий скрипт и средства управления им. В принципе, мы можем попробовать скормить ему что-нибудь уже сейчас, но осталась одна проблема - движок живет своей внутренней жизнью и мы все равно ничего не узнаем о результатах. Значит, мы должны обеспечить его средствами связи с нашим приложением.

Надеюсь, ни для кого не станет сюрпризом, что это "средство связи" является ничем иным, как еще одним объектом ;) Стало быть, настало применить на практике все теоритические знания о COM, потому что нам предстоит создать этот объект и реализовать все методы его интерфейсов. Интерфейсов будет два - **IActiveScriptSite** и **IActiveScriptSiteWindow**.

IActiveScriptSite для нас чрезвычайно важен т.к. помимо все прочего с его помощью движок информирует приложение о всех событиях, произошедших при обработке скрипта. Вот его методы в порядке следования в vtable:

• QueryInterface
• AddRef
• Release
• GetLCID
• GetItemInfo
• GetDocVersionString
• OnScriptTerminate
• OnStateChange
• OnScriptError
• OnEnterScript
• OnLeaveScript

А вот интерфейс IActiveScriptSiteWindow реализовывать строго говоря необязательно, но мы все же это сделаем. И вот почему - когда скрипт захочет как-нибудь взаимодействовать с GUI (например, отобразить обычный мессадж бокс) движку потребуется хэндл окна, которому этот месседж бокс принадлежит. Что он сделает? Запросит указатель на интерфейс IActiveScriptSiteWindow и с помощью метода GetWindow получит вожделенный hWnd. Если обнаружится, что объект не поддерживает IActiveScriptSiteWindow или что-нибудь в этом роде, то движок сообщит об ошибке и прекратит выполнение скрипта. Вот именно поэтому нам нужно реализовать этот интерфейс - чтобы увидеть результаты работы.

Реализация объекта.

Как я уже говорил спецификация COM не предусматривает никаких особых стандартов реализации объектов. Мы выберем самый простой путь - наш объект будет содержать:

- указатели на интерфейсы **IActiveScriptSite** и **IActiveScriptSiteWindow** ,
- счетчик ссылок (любой хороший объект должен содержать счетчик ссылок или хотя бы делать вид, что содержит ;)

Перво-наперво нужно определить, где это все будет храниться. Мы воспользуемся API LocalAlloc и выделим для него немного памяти.

```
    invoke LocalAlloc, LMEM_FIXED, 12
    mov [ActiveScriptSiteObject], eax
    mov [eax + 8], 1 ;Инициализируем счетчик ссылок
```

Теперь у нас есть объект - дело за интерфейсами. Сперва реализуем методы IUnknown:

```
  proc IActiveScriptSite_QueryInterface, pi, iid, ppvObject

    enter
                invoke IsEqualGUID, [iid], IID_IUnknown
    test eax, eax
    jnz .s_ok

    invoke IsEqualGUID, [iid], IID_IActiveScriptSite
    test eax, eax
    jnz .s_ok

    invoke IsEqualGUID, [iid], IID_IActiveScriptSiteWindow
    test eax, eax
    jnz .S_OK_W

  .NoInterface:   mov eax, E_NOINTERFACE

                  return
  .s_ok:          mov eax, [ActiveScriptSiteObject]
```

```
                mov eax, [eax]
                mov edx, [ppvObject]
                mov [edx], eax
                mov eax, S_OK

                return

.S_OK_W: mov eax, [ActiveScriptSiteObject]
                mov eax, [eax + 4]
                mov edx, [ppvObject]
                mov [edx], eax
                mov eax, S_OK
  return
```

Здесь мы с помощью API IsEqualGUID сверяем переданный нам GUID с эталонным и если они совпадают, то возвращаем указатель на соответствующий интерфейс. Пояснение здесь требуется только одно - так как наш объект не имеет самостоятельного **IUnknown**, то в ответ на идентификатор **00000000-0000-0000-0c00000000000046h** (GUID IUnknown) мы вернем указатель на IActiveScriptSite, что в данном случае одно и тоже.

```
proc IActiveScriptSite_AddRef, pi
               enter
                mov ecx, [ActiveScriptSiteObject]
                inc dword [ecx + 8]
                mov eax, [ecx + 8]
                return
```

Тут вообще комментариев не требуются ;)

```
proc IActiveScriptSite_Release, pi
               enter
                mov ecx, [ActiveScriptSiteObject]
               dec dword [ecx + 8]
               mov eax, [ecx + 8]
               test eax, eax
               jnz .ret_
               invoke LocalFree, ecx
.ret_:         return
```

Наконец, последний метод - Release - он уменьшает значение счетчика до тех пор, пока оно не станет равным 0, затем память освобождается с помощью LocalFree.

Теперь остается имплементировать следующие методы:

• GetLCID
• GetItemInfo
• GetDocVersionString
• OnScriptTerminate
• OnStateChange
• OnScriptError
• OnEnterScript
• OnLeaveScript
• GetWindow
• EnableModeless

Как видите, их довольно много, однако спешу вас обрадовать - не все из них требуется реализовывать полностью. Например, если вы не планируете какой-либо особой реакции своего приложения на начало исполнения скрипта можно вполне обойтись такой реализацией OnScriptEnter:

```
proc IActiveScriptSite_OnEnterScript, pi
                enter
                mov eax, S_OK
                return
```

Или например, если вы хотите доверить движку пользоваться для вывода текста установками по умолчанию, то можно **GetLCID** реализовать так:

```
proc IActiveScriptSite_GetLCID, pi, plcid
                enter
                mov eax, E_NOTIMPL
                return
```

На такой манер можно написать все методы IActiveScriptSite, что мы и сделаем.

```
proc IActiveScriptSite_GetItemInfo, pi, pstrName, dwReturnMask, ppunkItem, ppTypeInfo
               enter
                mov eax, TYPE_E_ELEMENTNOTFOUND
  return

proc IActiveScriptSite_GetDocVersionString, pi, pbstrVersionString
             enter
               mov eax, E_NOTIMPL
             return

proc IActiveScriptSite_OnScriptTerminate, pi, pvarResult, pexcepinfo
               enter
               mov eax, S_OK
               return

proc IActiveScriptSite_OnStateChange, pi, ssScriptState
               enter
                mov eax, S_OK
               return

proc IActiveScriptSite_OnEnterScript, pi
               enter
         mov eax, S_OK
               return

proc IActiveScriptSite_OnLeaveScript, pi
               enter
                mov eax, S_OK
               return
```

Только в метод OnScriptError (угадайте в какой ситуации он вызывается;)) мы кое-что добавим:

```
proc IActiveScriptSite_OnScriptError, pi, pase
               enter

  invoke MessageBoxA, 0, mess1, mess1_title, MB_OK + MB_ICONERROR
  mov eax, S_OK
            return
```

Теперь при обнаружение ошибки в скрипте будет появляться окно с соответствующим сообщением.

OK, с IActiveScriptSite вроде разобрались - настала очередь IActiveScriptSiteWindow. Тут будет проще - нужно написать все два метода.

```
proc IActiveScriptSiteWindow_GetWindow, pi, phwnd
  enter
  mov ecx, [phwnd]   ;Вернем hWnd нашего окна.
      mov eax, [hWnd]
      mov [ecx], eax
      mov eax, S_OK
      return

proc IActiveScriptSiteWindow_EnableModeless, pi, fEnable
      enter
      mov eax, S_OK
      return
```

Так как оба эти интерфейса принадлежат одному объекту, то пусть используют одни и те же стандартные методы. Обе Vtable разместим в сегменте данных, они будут такими:

```
pIActiveScriptSite_vtable   dd IActiveScriptSite_vtable
```

```
pIActiveScriptSiteWindow_vtable  dd IActiveScriptSiteWindow_vtable

IActiveScriptSite_vtable  dd IActiveScriptSite_QueryInterface
                          dd IActiveScriptSite_AddRef
                          dd IActiveScriptSite_Release
                          dd IActiveScriptSite_GetLCID
                          dd IActiveScriptSite_GetItemInfo
                          dd IActiveScriptSite_GetDocVersionString
                          dd IActiveScriptSite_OnScriptTerminate
                          dd IActiveScriptSite_OnStateChange
                          dd IActiveScriptSite_OnScriptError
                          dd IActiveScriptSite_OnEnterScript
                         dd IActiveScriptSite_OnLeaveScript

IActiveScriptSiteWindow_vtable dd IActiveScriptSite_QueryInterface
                         dd IActiveScriptSite_AddRef
                         dd IActiveScriptSite_Release
    dd IActiveScriptSiteWindow_GetWindow
    dd IActiveScriptSiteWindow_EnableModeless
```

Ну и напоследок поместим указатели на интерфейсы в выделенной для объекта памяти:

```
invoke LocalAlloc, LMEM_FIXED, 12
mov [ActiveScriptSiteObject], eax
mov [eax + 8], 1 ;Инициализируем счетчик ссылок
             mov dword [eax], pIActiveScriptSite_vtable
           mov dword [eax + 4], pIActiveScriptSiteWindow_vtable
```

Все! Наш объект готов, методы имплементированы, осталось только подготовить и запустить скрипт.

Подготовка.

Что же скрывается за словом "подготовить"? Ну во-первых, конечно же открыть файл со скриптом (если вы помните, его имя мы получаем при помощи GetOpenFileName). Открываем как обычно, используя API CreateFile:

```
invoke CreateFileA, [szFileName], GENERIC_READ, FILE_SHARE_READ, \
                    NULL, OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, NULL
mov [hFile], eax
```

Кроме этого нам нужен его размер:

```
invoke GetFileSize, eax, 0
mov [FileSize], eax
```

Вычислим сколько памяти нужно выделить (зачем умножать на 2 объясню чуть позже):

```
mov ecx, eax
inc eax
inc eax
shl eax, 1
push eax
add eax, ecx
mov [buffer_size], eax
```

Выделяем память при помощи VirtualAlloc:

```
invoke VirtualAlloc, 0, eax, MEM_COMMIT, PAGE_READWRITE
mov [pMemory], eax
```

Первые FileSize байт будут заняты содержимым файла, остальные пока свободны:

```
add eax, dword [FileSize]
inc eax
mov [pScript], eax

push eax
lea ecx, [esp]
invoke ReadFile, [hFile], [pMemory], [FileSize], ecx, NULL
pop ecx
```

За ненадобностью закрываем файл.

```
invoke CloseHandle, [hFile]

pop eax
```

Вот теперь раскажу зачем была выделена дополнительная память. Дело в том, что сейчас наш скрипт в памяти представляет собой обычную ASCII-последовательность, а VBScript engine оперирует со строками в формате Wide-Character (т.е. каждый символ занимает слово, а не байт). Поэтому наш скрипт еще предстоит конвертировать, что проще всего сделать с помощью MultiByteToWideChar:

```
invoke MultiByteToWideChar, 0, 0, [pMemory], -1, [pScript], eax
```

Запуск.

В этом разделе мы рассмотрим как инициализировать движок и передать ему скрипт.

Во-первых нужно увеличить значение счетчика нашего объекта, чтобы он не уничтожился раньше времени ;)

```
mov edx, [pIActiveScriptSite]
           mov edx, [edx]
           push [pIActiveScriptSite]
           call dword [edx + Addref]
```

Далее скажем IActiveScript, что использовать в качестве IActiveScriptSite необходимо реализованный нами интерфейс, для чего используем метод SetScriptSite, передав ему соответствующий указатель:

```
mov edx, [pIActiveScript]
           mov edx, [edx]
           push [pIActiveScriptSite]
           push [pIActiveScript]
           call dword [edx + 0ch]
```

Теперь инициализируем движок с помощью метода InitNew интерфейса IActiveScriptParse:

```
           mov edx, [pIActiveScriptParse]
           push edx
mov edx, [edx]
           call dword [edx + 0ch]
```

Загружаем скрипт, используя метод ParseScriptText. У этого метода куча аргументов, но в данном случае мы можем заменить их нулями. Хотя последний, конечно, не желательно, но я, лентяй этакий, поленился найти описание структуры EXCEPINFO и добавить его в заголовочный файл ;)))

```
mov edx, [pIActiveScriptParse]
           mov edx, [edx]
           mov eax, [pScript]
           sub ecx, ecx
push ecx
           push ecx
push ecx
           push ecx
push ecx
           push ecx
push ecx
           push ecx
push eax
push [pIActiveScriptParse]
call dword [edx + 14h]
```

После этого движоку будет загружен исходный код скрипта, но выполняться он не будет, до тех пор, пока движок не получит от нас соответствующей команды. Это объясняется тем, что движок сейчас пребывает не в том состоянии. Всего состояний 6, вот они:

```
    SCRIPTSTATE_UNINITIALIZED
    SCRIPTSTATE_INITIALIZED
    SCRIPTSTATE_STARTED
    SCRIPTSTATE_CONNECTED
    SCRIPTSTATE_DISCONNECTED
    SCRIPTSTATE_CLOSED
```

Собственно, названия говорят сами за себя, отмечу лишь, что для нас наиболее важно состояние SCRIPTSTATE_CONNECTED - именно оно заставит выполняться загруженный скрипт, поэтому, пользуясь методом SetScriptState, говорим движку, что пора бы уже исполнить чертов скрипт:

```
  mov edx, [pIActiveScript]
           mov edx, [edx]
  push SCRIPTSTATE_CONNECTED
  push [pIActiveScript]
       call dword [edx + 14h]

  mov edx, [pIActiveScriptSite]
           mov edx, [edx]
           push [pIActiveScriptSite]
           call dword [edx + Addref]

       invoke VirtualFree, [pMemory], [buffer_size], MEM_DECOMMIT
```

Ну вот мы заставили выполниться скрипт, теперь дело осталось за малым - правила хорошего тона требуют, чтобы мы по завершении работы прибрали за собой весь мусор. Давайте сделаем это в процедуре диалогового окна.

```
wmclose:  mov edx, [pIActiveScript]
            push edx
   mov edx, [edx]
            call dword [edx + 1Ch] ;Метод Close

              mov edx, [pIActiveScriptParse]
            push edx
   mov edx, [edx]
            call dword [edx + Release]

     mov edx, [pIActiveScript]
            push edx
   mov edx, [edx]
   push [pIActiveScript]
            call dword [edx + Release]

     mov edx, [pIActiveScriptSite]
            push edx
   mov edx, [edx]
            call dword [edx + Release]
```

Надеюсь, коментарии излишни ;)))?

Код приложения

```
;-------------------------------------------------------------------
; RunScript.asm
;-------------------------------------------------------------------

format PE GUI 4.0
entry start

include 'RunScript.inc'

section '.scrpt' code readable executable

start:       invoke GetModuleHandleA, 0
  mov [hInst], eax

  invoke DialogBoxParamA, eax, 1000, HWND_DESKTOP, DlgProc, 0

  invoke ExitProcess, 0
```

```
proc DlgProc, hwnd, msg, wparam, lparam
  enter
  cmp [msg], WM_INITDIALOG
  jz wminitdialog
  cmp [msg], WM_COMMAND
  jz wmcommand
  cmp [msg], WM_CLOSE
  jz wmclose
  sub eax, eax
  jmp finish

wminitdialog: mov eax, [hwnd]
  mov [hWnd], eax

  invoke VirtualAlloc, 0, MAX_PATH, MEM_COMMIT, PAGE_READWRITE
  mov [szFileName], eax

  call Create_Script_Engine

  jmp processed

wmcommand: cmp [wparam], BN_CLICKED shl 16 + 1001
  jnz processed

  mov eax, [szFileName]
  mov ecx, 257
@@:  mov byte [eax], 0
  inc eax
  dec ecx
  jnz @B

  push szFilter
  push szTitle
  push dword [hwnd]
  call GetFileName

  mov ecx, [szFileName]
  mov ecx, [ecx]
  jecxz processed

  call Get_Script
  test eax, eax
  jz processed

  push eax
  call Run_Script

  jmp processed

wmclose: invoke VirtualFree, [szFileName], MAX_PATH, MEM_DECOMMIT

  mov edx, [pIActiveScript]
             push edx
  mov edx, [edx]
             call dword [edx + Close]

               mov edx, [pIActiveScriptParse]
             push edx
  mov edx, [edx]
             call dword [edx + Release]

    mov edx, [pIActiveScript]
             push edx
  mov edx, [edx]
  push [pIActiveScript]
             call dword [edx + Release]

    mov edx, [pIActiveScriptSite]
             push edx
  mov edx, [edx]
             call dword [edx + Release]
```

```
            invoke EndDialog, [hwnd], 0

processed: sub eax, eax
  inc eax

finish:  return

proc Create_Script_Engine
  enter

            invoke CoInitialize, 0

            invoke CoCreateInstance, CLSID_VBScript, 0, \
          CLSCTX_INPROC_SERVER, IID_IActiveScript, pIActiveScript
            test eax, eax
            js error.create_instance

            invoke LocalAlloc, LMEM_FIXED, 12
            test eax, eax
            jz error.alloc

            mov [ActiveScriptSiteObject], eax

            mov dword [eax], pIActiveScriptSite_vtable
            mov dword [eax + 4], pIActiveScriptSiteWindow_vtable
  mov dword [eax + 8], 1

            mov edx, [pIActiveScript]
            mov edx, [edx]
            push pIActiveScriptParse
            push IID_IActiveScriptParse
            push [pIActiveScript]
            call dword [edx + QueryInterface]
            test eax, eax
            js .ok

            mov edx, pIActiveScriptSite_vtable
  mov [pIActiveScriptSite], edx

.ok:        sub eax, eax
            return

error:

.alloc:  mov eax, E_OUTOFMEMORY
  return

.create_instance:

            sub eax, eax
            dec eax
            return

proc Get_Script
  enter

  invoke CreateFileA, [szFileName], GENERIC_READ, FILE_SHARE_READ, \
                      NULL, OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, NULL
  test eax, eax
  jz LoadScript.finish

  mov [hFile], eax

  invoke GetFileSize, eax, 0

  mov [FileSize], eax

  mov ecx, eax
  inc eax
  inc eax
  shl eax, 1
```

```
  push eax
  add eax, ecx
  mov [buffer_size], eax

  invoke VirtualAlloc, 0, eax, MEM_COMMIT, PAGE_READWRITE
  mov [pMemory], eax

  add eax, dword [FileSize]
  inc eax
  mov [pScript], eax

  push eax
  lea ecx, [esp]
  invoke ReadFile, [hFile], [pMemory], [FileSize], ecx, NULL
  pop ecx

  invoke CloseHandle, [hFile]

  pop eax

  invoke MultiByteToWideChar, 0, 0, [pMemory], -1, [pScript], eax

      mov eax, [pScript]

.finish: return

proc Run_Script, lpScript

  enter

  mov edx, [pIActiveScriptSite]
            mov edx, [edx]
            push [pIActiveScriptSite]
            call dword [edx + Addref]

            mov edx, [pIActiveScript]
            mov edx, [edx]
            push [pIActiveScriptSite]
            push [pIActiveScript]
            call dword [edx + SetScriptSite]

            mov edx, [pIActiveScriptParse]
            mov edx, [edx]
            push [pIActiveScriptParse]
            call dword [edx + InitNew]

            mov edx, [pIActiveScriptParse]
            mov edx, [edx]
            mov eax, [lpScript]
            sub ecx, ecx
  push ecx
            push ecx
  push ecx
            push ecx
  push ecx
            push ecx
  push ecx
            push ecx
  push eax
  push [pIActiveScriptParse]
  call dword [edx + ParseScriptText]

  mov edx, [pIActiveScript]
            mov edx, [edx]
  push SCRIPTSTATE_CONNECTED
  push [pIActiveScript]
       call dword [edx + SetScriptState]

       mov edx, [pIActiveScriptSite]
           mov edx, [edx]
           push [pIActiveScriptSite]
```

```
             call dword [edx + Addref]

         invoke VirtualFree, [pMemory], [buffer_size], MEM_DECOMMIT

  return

proc IActiveScriptSite_GetLCID, pi, plcid
                enter
                mov eax, E_NOTIMPL
                return

proc IActiveScriptSite_GetItemInfo, pi, pstrName, dwReturnMask, ppunkItem, ppTypeInfo
                enter
                mov eax, TYPE_E_ELEMENTNOTFOUND
  return

proc IActiveScriptSite_GetDocVersionString, pi, pbstrVersionString
                enter
                mov eax, E_NOTIMPL
                return

proc IActiveScriptSite_OnScriptTerminate, pi, pvarResult, pexcepinfo
               enter
               mov eax, S_OK
               return

proc IActiveScriptSite_OnStateChange, pi, ssScriptState
               enter
                mov eax, S_OK
            return

proc IActiveScriptSite_OnScriptError, pi, pase
           enter

               invoke MessageBoxA, 0, mess1, mess1_title, MB_OK + MB_ICONERROR
  mov eax, S_OK
               return

proc IActiveScriptSite_OnEnterScript, pi
               enter
               mov eax, S_OK
              return

proc IActiveScriptSite_OnLeaveScript, pi
               enter
                mov eax, S_OK
            return

proc IActiveScriptSite_AddRef, pi
               enter
               mov ecx, [ActiveScriptSiteObject]
               inc dword [ecx + 8]
               mov eax, [ecx + 8]
               return

proc IActiveScriptSite_Release, pi
               enter
                mov ecx, [ActiveScriptSiteObject]
               dec dword [ecx + 8]
               mov eax, [ecx + 8]
              test eax, eax
               jnz .ret_
                invoke LocalFree, ecx
.ret_:         return

proc IActiveScriptSite_QueryInterface, pi, iid, ppvObject

                enter

                invoke IsEqualGUID, [iid], IID_IUnknown
                test eax, eax
```

```
                jnz .s_ok

                invoke IsEqualGUID, [iid], IID_IActiveScriptSite
                test eax, eax
                jnz .s_ok

                invoke IsEqualGUID, [iid], IID_IActiveScriptSiteWindow
                test eax, eax
                jnz .S_OK_W

.NoInterface:   mov eax, E_NOINTERFACE

                return

.s_ok:          mov eax, [ActiveScriptSiteObject]
                mov eax, [eax]
                mov edx, [ppvObject]
                mov [edx], eax
                mov eax, S_OK

                return

.S_OK_W: mov eax, [ActiveScriptSiteObject]
                mov eax, [eax + 4]
               mov edx, [ppvObject]
               mov [edx], eax
               mov eax, S_OK
  return

proc IActiveScriptSiteWindow_GetWindow, pi, phwnd
  enter
  mov ecx, [phwnd]
      mov eax, [hWnd]
      mov [ecx], eax
      mov eax, S_OK
      return

proc IActiveScriptSiteWindow_EnableModeless, pi, fEnable
      enter
      mov eax, S_OK
      return

proc GetFileName, hParent, lpTitle, lpFilter

  enter
  mov eax, [hParent]
      mov [ofn.hwndOwner], eax
     mov eax, [hInst]
     mov [ofn.hInstance], eax
     mov eax, [lpFilter]
      mov [ofn.lpstrFilter], eax
      mov eax, [szFileName]
      mov [ofn.lpstrFile], eax
      mov [ofn.nMaxFile], MAX_PATH
      mov eax, [lpTitle]
      mov [ofn.lpstrTitle], eax
      mov [ofn.Flags], OFN_EXPLORER or OFN_FILEMUSTEXIST or OFN_LONGNAMES

  invoke GetOpenFileName, ofn

  return

section '.data' data readable writeable

ofn OPENFILENAMEA

mess1 db "Cant't parse it!", 0
mess1_title db 'script engine', 0

szFilter db 'VBScripts', 0, '*.vbs', 0, \
     'All files (heh... you can try...)', 0, '*.*', 0, 0
```

```
szTitle db 'Just open fucking file...', 0
szFileName dd 0

hInst dd 0

ActiveScriptSiteObject dd 0

hWnd dd 0
hFile dd 0
FileSize dd 0
buffer_size dd 0
pMemory dd 0
pScript dd 0

pIActiveScript dd 0
pIActiveScriptParse dd 0
pIActiveScriptSite dd 0

CLSID_VBScript GUID 0b54f3741h, 5b07h, 11cfh, 0a4h, 0b0h, 0, 0aah, 0, 4ah, 55h, 0e8h
IID_IActiveScript GUID 0bb1a2ae1h, 0a4f9h, 11cfh, 8fh, 20h, 0, 80h, 5fh, 2ch, 0d0h, 64h
IID_IActiveScriptParse GUID 0bb1a2ae2h, 0a4f9h, 11cfh, 8fh, 20h, 0, 80h, 5fh, 2ch, 0d0h, 64h
IID_IActiveScriptSite GUID 0d57d7817h, 0e9b7h, 04a82h, 85h, 74h, 01h, 0d0h, 0f9h, 3dh, 61h, 70h
IID_IActiveScriptSiteWindow GUID 0d10f6761h, 083e9h, 011cfh, 8fh, 20h, 0, 80h, 5fh, 2ch, 0d0h, 64h
IID_IUnknown GUID 0, 0, 0, 0ch, 0, 0, 0, 0, 0, 0, 46h

pIActiveScriptSite_vtable dd IActiveScriptSite_vtable
pIActiveScriptSiteWindow_vtable dd IActiveScriptSiteWindow_vtable

IActiveScriptSite_vtable dd IActiveScriptSite_QueryInterface
                         dd IActiveScriptSite_AddRef
                         dd IActiveScriptSite_Release
                         dd IActiveScriptSite_GetLCID
                         dd IActiveScriptSite_GetItemInfo
                         dd IActiveScriptSite_GetDocVersionString
                         dd IActiveScriptSite_OnScriptTerminate
                         dd IActiveScriptSite_OnStateChange
                         dd IActiveScriptSite_OnScriptError
                         dd IActiveScriptSite_OnEnterScript
                         dd IActiveScriptSite_OnLeaveScript

IActiveScriptSiteWindow_vtable dd IActiveScriptSite_QueryInterface
                         dd IActiveScriptSite_AddRef
                         dd IActiveScriptSite_Release
    dd IActiveScriptSiteWindow_GetWindow
    dd IActiveScriptSiteWindow_EnableModeless

QueryInterface   =  0
Addref           =  4
Release          =  8
Close    =  1ch
SetScriptSite   =  0ch
InitNew   = 0ch
ParseScriptText   =  14h
SetScriptState   =  14h

section '.idata' import data readable writeable
library kernel32, 'kernel32.dll', \
 user32, 'user32.dll', \
        ole32, 'ole32.dll', \
        comdlg32, 'comdlg32.dll'

import kernel32, ExitProcess, 'ExitProcess', \
     LocalAlloc, 'LocalAlloc', \
          LocalFree, 'LocalFree', \
          GetModuleHandleA, 'GetModuleHandleA', \
          CreateFileA, 'CreateFileA', \
          CloseHandle, 'CloseHandle', \
          ReadFile, 'ReadFile', \
          GetFileSize, 'GetFileSize', \
          MultiByteToWideChar, 'MultiByteToWideChar', \
          VirtualAlloc, 'VirtualAlloc', \
```

```
          VirtualFree, 'VirtualFree'

import ole32, CoInitialize, 'CoInitialize', \
              CoCreateInstance, 'CoCreateInstance', \
              CoUninitialize, 'CoUninitialize', \
              IsEqualGUID, 'IsEqualGUID'

import user32, DialogBoxParamA, 'DialogBoxParamA', \
        EndDialog, 'EndDialog', \
        MessageBoxA, 'MessageBoxA'

import comdlg32, GetOpenFileName, 'GetOpenFileNameA'

section '.rsrc' resource data readable

directory RT_DIALOG, dialogs

resource dialogs, 1000, LANG_ENGLISH+SUBLANG_DEFAULT, main_dlg

  dialog main_dlg, 'RunScript', 70, 70, 70, 50, WS_CAPTION+WS_POPUP+WS_SYSMENU+DS_MODALFRAME
  dialogitem 'BUTTON', 'Execute Script!', 1001, 8, 15, 55, 15, WS_VISIBLE+WS_TABSTOP+BS_PUSHBUTTON

enddialog

;-----------------------------------------------------------------------
; RunScript.inc
;-----------------------------------------------------------------------

include 'include\win32ax.inc'

CLSCTX_INPROC_SERVER   equ 1
CLSCTX_INPROC_HANDLER   equ 2
CLSCTX_LOCAL_SERVER       equ 4
CLSCTX_INPROC_SERVER16   equ 8
CLSCTX_REMOTE_SERVER   equ 10h
CLSCTX_INPROC_HANDLER16  equ 20h
CLSCTX_INPROC_SERVERX86  equ 40h
CLSCTX_INPROC_HANDLERX86  equ 80h
CLSCTX_ESERVER_HANDLER   equ 100h

E_NOTIMPL    equ 80004001h
E_NOINTERFACE                 equ 80004002h

TYPE_E_ELEMENTNOTFOUND       equ 8002802Bh

S_OK                          equ 0

LMEM_FIXED                    equ 0h

E_OUTOFMEMORY                 equ 8007000Eh

SCRIPTSTATE_UNINITIALIZED   equ 0h
SCRIPTSTATE_STARTED         equ 1h
SCRIPTSTATE_CONNECTED         equ 2h
SCRIPTSTATE_DISCONNECTED    equ 3h
SCRIPTSTATE_CLOSED          equ 4h
SCRIPTSTATE_INITIALIZED     equ 5h

struc GUID dd1, dw1, dw2, db1, db2, db3, db4, db5, db6, db7, db8
{
.dd1 dd dd1
.dw1 dw dw1
.dw2 dw dw2
.db1 db db1
.db2 db db2
.db3 db db3
.db4 db db4
.db5 db db5
.db6 db db6
.db7 db db7
.db8 db db8
```

```
}

MAX_PATH equ 260

struc OPENFILENAMEA
{
.lStructSize   dd 4ch
.hwndOwner          dd 0
.hInstance          dd 0
.lpstrFilter        dd 0
.lpstrCustomFilter  dd 0
.nMaxCustFilter     dd 0
.nFilterIndex       dd 0
.lpstrFile          dd 0
.nMaxFile           dd 0
.lpstrFileTitle     dd 0
.nMaxFileTitle      dd 0
.lpstrInitialDir    dd 0
.lpstrTitle         dd 0
.Flags              dd 0
.nFileOffset        dw 0
.nFileExtension     dw 0
.lpstrDefExt        dd 0
.lCustData          dd 0
.lpfnHook           dd 0
.lpTemplateName     dd 0
}
struct OPENFILENAMEA
```

На сегодня все :).

Файл (находится в архиве `wasm_files.zip`: `files/0299/runscript.zip`) к статье.

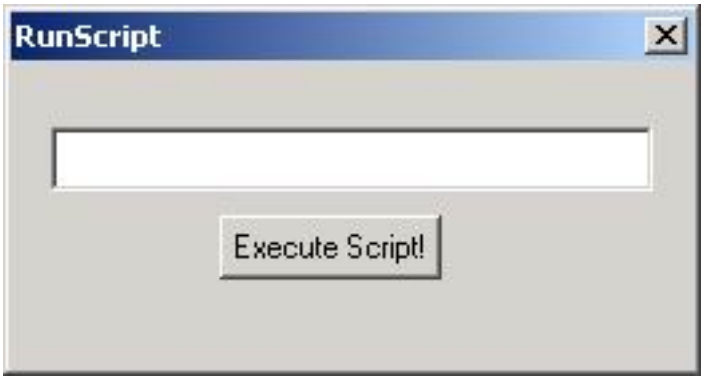
Введение в использование скриптовых движков: Часть 2

Автор: Hangatyr

Дата: 2004-02-24

Введение

Если вы помните, в предыдущей статье было рассказано об основах управления скриптовыми движками, в частности о том, как заставить выполняться простейший скрипт. В том примере было мало интересного - скрипт жил сам по себе и, отработав, просто возвращал управление основной программе. Сейчас настало время рассмотреть более полезные техники, а именно, как передать скрипту какие-нибудь параметры и наоборот - получить от него информацию (например, о результатах работы). В качестве иллюстрации рассмотрим такой пример:



Сделаем так, чтобы текст в EditBox’е можно было изменять из скрипта.

В прошлый раз я говорил о том, что необходимым условием для использования COM-объекта является знание структуры его интерфейсов. Однако далеко не всегда такое возможно, как например в нашем случае. Например, так будет выглядеть скрипт, устанавливающий содержимое контрола Edit равным "some text":

```
Edit1.Text = "some sext"
```

Edit1 - это имя нашего объекта, а Text - свойство, подлежащее установке. Ясно, что никаких данных о структуре интерфейса из этого извлечь нельзя, поэтому нужно располагать некоторым средством, позволяющим вызывать методы по их именам. Такое средство есть, в COM оно известно, как "позднее связывание" (late binding).

LATE BINDING.

Скажу сразу, что здесь мы рассмотрим только тот минимум, что необходим для работы. Как уже было сказано выше, позднее связывание применяется в тех случаях, когда по каким-либо причинам невозможно (или неудобно) предоставить информацию о структуре интерфейса.

Выходом является использование интерфейса **IDispatch** , который возволяет вызвать нужную процедуру по ее имени. Он содержит следующие методы:

• GetTypeInfoCount
• GetTypeInfo
• GetIDsOfNames
• Invoke

(нет необходимости говорить о том, что **IDispatch** тоже наследуется от **IUnknown**).

Из этих методов нам интересны два последних, обычно их использование выглядит так: сначала методу GetIDsOfNames передается массив с адресами имен нужного метода (свойства) и его параметров, он этот массив просматривает и устанавливает соответствие между именем и некоторым числом, представляющим собой уникальный (в пределах интефейса) ID (обычно говорят “DISPID”) данного элемента. Далее этот DISPID передается методу Invoke, который и производит вызов процедуры.

Имя объекта

С именем объекта все гораздо проще – достаточно указать движку, что за именем “Edit1” скрывается объект.

Делается это при помощи вызова IActiveScript::AddNamedItem.

HRESULT AddNamedItem(LPCOLESTR pstrName, DWORD dwFlags);

pstrName– указатель на имя в формате WideCharacter. **dwFlags** может принимать разные значения, в зависимости от поставленной задачи, мы укажем флаг SCRIPTITEM_ISVISIBLE, показывающий, что **pstrName** – это имя объекта и мы предоставляем средства для доступа к его методам и свойствам.

```
mov edx, [pIActiveScript]
mov edx, [edx]
push SCRIPTITEM_ISVISIBLE
push Item_name
push [pIActiveScript]
call dword [edx + AddNamedItem]
```

Теперь, когда движок встретит конструкцию Edit1.Method он вызовет метод IActiveScriptSite::GetItemInfo, чтобы получить интерфейс IDispatch, который контролирует данный объект.

*HRESULT GetItemInfo(LPCOLESTR pstrName, DWORD dwReturnMask, IUnknown ppunkItem, ITypeInfo ppTypeInfo);***pstrName**– указатель на имя объекта. **dwReturnMask** – показывает, какая именно информация запрашивается. Может принимать значения:

• SCRIPTINFO_IUNKNOWN – запрашивается **IUnknown**
• SCRIPTINFO_ITYPEINFO – запрашивается **ITypeInfo**

ppunkItem, ppTypeInfo– адреса переменных, в которые будут помещены указатели на интерфейсы. Здесь есть один момент, который заслуживает внимания – дело в том, что вышеописанный **IDispatch**– не едиственный способ осуществить связывание имен. В качестве альтернативы может использоваться интерфейс **ITypeInfo**, но мы не будем его реализовывать и все сделаем через **IDispatch**, поэтому если наш метод GetItemInfo получает параметр **dwReturnMask** с установленным SCRIPTINFO_ITYPEINFO, он должен будет вернуть 0 в **ppTypeInfo**.

```
proc IActiveScriptSite_GetItemInfo, pi, pstrName, dwReturnMask, ppunkItem, ppTypeInfo
  enter

  ;Сравниваем переданное имя с “Edit1”.
  invoke lstrcmpW, [pstrName], Item_name
  test eax, eax
  jz @F

  mov eax, TYPE_E_ELEMENTNOTFOUND
  return

@@:  mov eax, [dwReturnMask]

  cmp eax, SCRIPTINFO_IUNKNOWN
  jz .IUnknown

  cmp eax, SCRIPTINFO_ALL_FLAGS
```

```
  jz .IUnknown

  cmp eax, SCRIPTINFO_ITYPEINFO
  jz .ITypeInfo

  mov eax, E_INVALIDARG
  return

.IUnknown:
  mov eax, [ppunkItem]
         mov edx, IDispatch
         mov [eax], edx

         mov edx, [edx]
         push IDispatch
         call dword [edx + Addref]

.ITypeInfo: mov eax, [ppTypeInfo]
         test eax, eax
         jz @F

         and dword [eax], 0

@@:      mov eax, S_OK
         return
```

Теперь, чтобы вызвать нужный метод движок предпримет следующие действия:

1. Использует метод **QueryInterface**, полученного **IUnknown** для получения указателя на **IDispatch**.
2. С помощью **GetIDsOfNames** преобразует имена методов (свойств) в их идентификаторы.
3. Вызовет нужный код с помощью **Invoke**

Следовательно, первое, что нам нужно сделать – подправить метод IUnknown::QueryInterface :

```
proc IActiveScriptSite_QueryInterface, pi, iid, ppvObject
  enter
  invoke IsEqualGUID, [iid], IID_IUnknown
  test eax, eax
  jnz .s_ok_unk

  invoke IsEqualGUID, [iid], IID_IActiveScriptSite
  test eax, eax
  jnz .s_ok_unk

  invoke IsEqualGUID, [iid], IID_IActiveScriptSiteWindow
  test eax, eax
  jnz .s_ok_acw

  invoke IsEqualGUID, [iid], IID_IDispatch
  test eax, eax
  jnz .s_ok_dsp

.NoInterface: mov eax, E_NOINTERFACE
  return

.s_ok:  mov edx, [pIActiveScriptSite]
               mov edx, [edx]
  push [pIActiveScriptSite]
  call dword [edx + Addref]

  mov eax, S_OK
  return

.s_ok_unk: mov eax, [ActiveScriptSiteObject]
  mov eax, [eax]
  mov edx, [ppvObject]
  mov [edx], eax
  jmp .s_ok

.s_ok_acw: mov eax, [ActiveScriptSiteObject]
  mov eax, [eax + 4]
```

```
  mov edx, [ppvObject]
  mov [edx], eax
  jmp .s_ok

.s_ok_dsp: ;если iid == IDispath, то вернем указатель на него.

  mov eax, [ActiveScriptSiteObject]
  mov eax, IDispatch
  mov edx, [ppvObject]
  mov [edx], eax
  jmp .s_ok
```

Теперь QueryInterface способен воспринимать GUID IDispatch *(00020400-0000-0000-C000-000000000046)* , а сам IDispatch будет выглядеть так:

```
IDispatch      dd IDispatch_vtable

;первые 3 – методы IUnknown, общие для всех интерфейсов нашего
объекта.

IDispatch_vtable dd IActiveScriptSite_QueryInterface
   dd IActiveScriptSite_AddRef
   dd IActiveScriptSite_Release

   dd IDispatch_GetTypeInfoCount
   dd IDispatch_GetTypeInfo
   dd IDispatch_GetIDsOfNames
   dd IDispatch_Invoke
```

Методы GetTypeInfoCount и GetTypeInfo мы здесь рассматривать не будем – заменим их чем-нибудь вроде этого:

```
proc IDispatch_GetTypeInfoCount, pi, pctinfo
   enter
   mov eax, [pctinfo]
   mov dword [eax], 0
   mov eax, S_OK
   return

proc IDispatch_GetTypeInfo, pi, iTInfo, lcid, ppTInfo
   enter
   mov eax, DISP_E_BADINDEX
   return
```

А вот без Invoke и GetIDsOfNames не обойтись.

HRESULT GetIDsOfNames(REFIID riid, OLECHAR FAR FAR* rgszNames, unsigned int cNames, LCID lcid, DISPID FAR* rgDispId);*

riid– reserved (всегда 0). **rgszNames**– указатель на массив указателей на строки, содержащие имена (первый представляет собой имя метода/свойства, следующие - имена параметров). **cNames**– количество элементов массива. **rgDispId**– указатель на массив, в который будут помещены ID, соответствующие именам из **rgszNames**. Если имя не опознано, то в соответствующий элемент массива **rgDispId** должно быть помещено значение **-1**. **lcid** нам не понадобится, т.к. предполагаем, что имена будут только на английском.

Вот примерная (и не совсем корректная) реализация:

```
proc IDispatch_GetIDsOfNames, pi, riid, rgszNames, cNames, lcid, rgDispId
  enter

  pushad

  mov esi, [rgszNames]
  mov ebx, [rgDispId]

  lodsd

  invoke lstrcmpW, eax, szSetText
```

```
  test eax, eax
  jnz @F

  ;Пусть DISPID будет равен 1
  mov dword [ebx], 1

  popad
  mov eax, S_OK
  return

@@:  or dword [ebx], -1
  popad

  ;Было передано имя несуществующего элемента.
  mov eax, DISP_E_UNKNOWNNAME
  return
```

Позволю себе сделать небольшое отступление, перед тем, как привести исходный код метода **Invoke**. Дело в том, что **IDispatch** представляет из себя только диспетчер, который вызывает нужную процедуру в зависимости от переданного DISPID, поэтому нам нужно написать код, устанавливающий текст в EditBox'е (правда, текст этот надо сначала привести в божеский вид):

```
proc SetText, pNewText
  enter
  pushad
  mov ebx, [pNewText]

  ;определим необходимое кол-во памяти.
  invoke WideCharToMultiByte, CP_ACP, 0, ebx, -1, 0, 0, 0, 0

  push MEM_DECOMMIT
  push eax

  invoke VirtualAlloc, 0, eax, MEM_COMMIT, PAGE_READWRITE

  pop ecx
  push ecx
  push eax

  invoke WideCharToMultiByte, CP_ACP, 0, ebx, -1, eax, ecx, 0, 0

  invoke GetDlgItem, [hWnd], 1002
  mov ecx, [esp]
  invoke SetWindowTextA, eax, ecx

  invoke VirtualFree

  popad
  mov eax, S_OK
  return
```

Теперь займемся самим методом IDispatch::Invoke

HRESULT Invoke(DISPID dispIdMember, REFIID riid, LCID lcid, WORD wFlags, DISPPARAMS FAR pDispParams, VARIANT FAR* pVarResult, EXCEPINFO FAR* pExcepInfo, unsigned int FAR* puArgErr);*

dispIdMember– вызываемый ID (в нашем случае 1). **riid** - Reserved. Должен быть IID_NULL. **wFlags** может принимать значения:

• DISPATCH_METHOD
• DISPATCH_PROPERTYGET
• DISPATCH_PROPERTYPUT (в нашем случае будет именно это значение)

pDispParams– указатель на структуру **DISPPARAMS** , через которую передаются параметры.

```
typedef struct FARSTRUCT tagDISPPARAMS{
VARIANTARG FAR* rgvarg;
   DISPID FAR* rgdispidNamedArgs;
   Unsigned int cArgs;
```

```
   Unsigned int cNamedArgs;
} DISPPARAMS;
```

Так как параметр нам передается всего один (строка), то достаточно разобрать только **rgvarg**. Единственная проблема – элемент этого массива – указатель не на саму строку, а на структуру VARIANTARG:

```
typedef struct FARSTRUCT tagVARIANT VARIANTARG;

typedef struct tagVARIANT  {
   VARTYPE vt;
   unsigned short wReserved1;
   unsigned short wReserved2;
   unsigned short wReserved3;
   union {
      Byte   bVal;   // VT_UI1.
      Short   iVal;   // VT_I2.
      long   lVal;   // VT_I4.
      float   fltVal;   // VT_R4.
      double   dblVal;   // VT_R8.
      VARIANT_BOOL  boolVal;  // VT_BOOL.
      SCODE   scode;   // VT_ERROR.
      CY   cyVal;   // VT_CY.
      DATE   date;   // VT_DATE.
      BSTR   bstrVal;  // VT_BSTR.
      DECIMAL   FAR* pdecVal;  //VT_BYREF|VT_DECIMAL.
      IUnknown   FAR* punkVal;  // VT_UNKNOWN.
      IDispatch   FAR* pdispVal;  //VT_DISPATCH.
      SAFEARRAY   FAR* parray;  // VT_ARRAY|*.
      Byte   FAR* pbVal;  //VT_BYREF|VT_UI1.
      short   FAR* piVal;  //VT_BYREF|VT_I2.
      long   FAR* plVal;  //VT_BYREF|VT_I4.
      float   FAR* pfltVal;  //VT_BYREF|VT_R4.
      double   FAR* pdblVal;  //VT_BYREF|VT_R8.
      VARIANT_BOOL  FAR* pboolVal;  //VT_BYREF|VT_BOOL.
      SCODE   FAR* pscode;  //VT_BYREF|VT_ERROR.
      CY   FAR* pcyVal;  //VT_BYREF|VT_CY.
      DATE   FAR* pdate;  //VT_BYREF|VT_DATE.
      BSTR   FAR* pbstrVal;  //VT_BYREF|VT_BSTR.
      IUnknown   FAR* FAR* ppunkVal; //VT_BYREF|VT_UNKNOWN.
      IDispatch   FAR* FAR* ppdispVal; //VT_BYREF|VT_DISPATCH.
      SAFEARRAY   FAR* FAR* pparray; // VT_ARRAY|*.
      VARIANT   FAR* pvarVal;  //VT_BYREF|VT_VARIANT.
      void   FAR* byref;  // GenericByRef.
      char   cVal;   // VT_I1.
      unsigned short  uiVal;   // VT_UI2.
      unsigned long  ulVal;   // VT_UI4.
      int   intVal;   // VT_INT.
      unsigned int  uintVal;  // VT_UINT.
      char FAR *  pcVal;   //VT_BYREF|VT_I1.
      unsigned short FAR * puiVal;   //VT_BYREF|VT_UI2.
      unsigned long FAR * pulVal;   //VT_BYREF|VT_UI4.
      int FAR *   pintVal;  //VT_BYREF|VT_INT.
      unsigned int FAR * puintVal;  //VT_BYREF|VT_UINT.
   };
};
```

Как видите, сам указатель находится по смещению +08 от начала структуры.

Вот теперь можно привести код Invoke:

```
proc IDispatch_Invoke, pi, dispIdMember, riid, lcid, wFlags, pDispParams, \
     pVarResult, pExcepInfo, puArgErr
 enter

 mov eax, [dispIdMember]
 dec eax
 jnz IDispatch_Invoke.notfound

 mov edx, [pDispParams]
 mov edx, [edx]
```

```
 mov edx, [edx + 8]

 push edx
 call SetText

 mov eax, S_OK
 return

.notfound:
 mov eax, DISP_E_MEMBERNOTFOUND
 return
```

Код приложения

```
;-----------------------------------------------------------------------
; RunScript.asm
;-----------------------------------------------------------------------

format PE GUI 4.0
entry start

include 'RunScript.inc'

section '.scrpt' code readable executable

start:       invoke GetModuleHandleA, 0
  mov [hInst], eax
  invoke DialogBoxParamA, eax, 1000, HWND_DESKTOP, DlgProc, 0
  invoke ExitProcess, 0

proc DlgProc, hwnd, msg, wparam, lparam
  enter
  cmp [msg], WM_INITDIALOG
  jz wminitdialog

  cmp [msg], WM_COMMAND
  jz wmcommand

  cmp [msg], WM_CLOSE
  jz wmclose

  sub eax, eax
  jmp finish

wminitdialog: mov eax, [hwnd]
  mov [hWnd], eax

  invoke VirtualAlloc, 0, MAX_PATH, MEM_COMMIT, PAGE_READWRITE
  mov [szFileName], eax

  call StartScriptEngine

  jmp processed

wmcommand: cmp [wparam], BN_CLICKED shl 16 + 1001
  jnz processed

  mov eax, [szFileName]
  mov ecx, 257
@@:  mov byte [eax], 0
  inc eax
  dec ecx
  jnz @B

  push szFilter
  push szTitle
  push dword [hwnd]
  call GetFileName

  mov ecx, [szFileName]
  mov ecx, [ecx]
```

```
  jecxz processed

  call LoadScript
  test eax, eax
  jz processed

  push eax
  call RunScript

  jmp processed

wmclose: invoke VirtualFree, [szFileName], MAX_PATH, MEM_DECOMMIT

  mov edx, [pIActiveScript]
             push edx
  mov edx, [edx]
             call dword [edx + Close]

               mov edx, [pIActiveScriptParse]
             push edx
  mov edx, [edx]
             call dword [edx + Release]

    mov edx, [pIActiveScript]
             push edx
  mov edx, [edx]
  push [pIActiveScript]
             call dword [edx + Release]

    mov edx, [pIActiveScriptSite]
             push edx
  mov edx, [edx]
             call dword [edx + Release]

             invoke EndDialog, [hwnd], 0

processed: sub eax, eax
  inc eax

finish:  return

proc StartScriptEngine

  enter

             invoke CoInitialize, 0

             invoke CoCreateInstance, CLSID_VBScript, 0, CLSCTX_INPROC_SERVER, \
      IID_IActiveScript, pIActiveScript
             test eax, eax
             js error.create_instance

             invoke LocalAlloc, LMEM_FIXED, 16
             test eax, eax
             jz error.alloc

             mov [ActiveScriptSiteObject], eax

             mov dword [eax], IActiveScriptSite
             mov dword [eax + 04], IActiveScriptSiteWindow
             mov dword [eax + 08], IDispatch
  mov dword [eax + 12], 1

             mov edx, [pIActiveScript]
             mov edx, [edx]
             push pIActiveScriptParse
             push IID_IActiveScriptParse
             push [pIActiveScript]
             call dword [edx + QueryInterface]
             test eax, eax
             js .ok
```

```
            mov edx, IActiveScriptSite
  mov [pIActiveScriptSite], edx

.ok:        sub eax, eax
            return

error:

.alloc:  mov eax, E_OUTOFMEMORY
  return

.create_instance:

            sub eax, eax
            dec eax
            return

proc LoadScript
  enter

  invoke CreateFileA, [szFileName], GENERIC_READ, FILE_SHARE_READ, NULL, \
      OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, NULL
  test eax, eax
  jz LoadScript.finish

  mov [hFile], eax

  invoke GetFileSize, eax, 0

  mov [FileSize], eax

  mov ecx, eax
  inc eax
  inc eax
  shl eax, 1
  push eax
  add eax, ecx
  mov [buffer_size], eax

  invoke VirtualAlloc, 0, eax, MEM_COMMIT, PAGE_READWRITE
  mov [pMemory], eax

  add eax, dword [FileSize]
  inc eax
  mov [pScript], eax

  push eax
  lea ecx, [esp]
  invoke ReadFile, [hFile], [pMemory], [FileSize], ecx, NULL
  pop ecx

  invoke CloseHandle, [hFile]

  pop eax

  invoke MultiByteToWideChar, 0, 0, [pMemory], -1, [pScript], eax

      mov eax, [pScript]

.finish: return

proc RunScript, lpScript

  enter

  mov edx, [pIActiveScriptSite]
            mov edx, [edx]
            push [pIActiveScriptSite]
            call dword [edx + Addref]

            mov edx, [pIActiveScript]
```

```
             mov edx, [edx]
            push [pIActiveScriptSite]
             push [pIActiveScript]
             call dword [edx + SetScriptSite]

             mov edx, [pIActiveScriptParse]
             mov edx, [edx]
             push [pIActiveScriptParse]
             call dword [edx + InitNew]

             mov edx, [pIActiveScript]
             mov edx, [edx]
             push SCRIPTITEM_ISVISIBLE
             push Item_name
             push [pIActiveScript]
             call dword [edx + AddNamedItem]

            mov edx, [pIActiveScriptParse]
            mov edx, [edx]
             mov eax, [lpScript]
             sub ecx, ecx
  push ecx
             push ecx
  push ecx
             push ecx
  push ecx
             push ecx
  push ecx
           push ecx
  push eax
  push [pIActiveScriptParse]
  call dword [edx + ParseScriptText]

  mov edx, [pIActiveScript]
             mov edx, [edx]
  push SCRIPTSTATE_CONNECTED
  push [pIActiveScript]
         call dword [edx + SetScriptState]

  invoke VirtualFree, [pMemory], [buffer_size], MEM_DECOMMIT

  return

proc IActiveScriptSite_GetLCID, pi, plcid
                enter
                mov eax, E_NOTIMPL
                return

proc IActiveScriptSite_GetItemInfo, pi, pstrName, dwReturnMask, ppunkItem, ppTypeInfo
                enter
               invoke lstrcmpW, [pstrName], Item_name
  test eax, eax
  jz @F

  mov eax, TYPE_E_ELEMENTNOTFOUND
  return

@@:  mov eax, [dwReturnMask]

  cmp eax, SCRIPTINFO_IUNKNOWN
  jz .IUnknown

  cmp eax, SCRIPTINFO_ALL_FLAGS
  jz .IUnknown

  cmp eax, SCRIPTINFO_ITYPEINFO
  jz .ITypeInfo

  mov eax, E_INVALIDARG
  return
.IUnknown: mov eax, [ppunkItem]
```

```
  mov edx, IDispatch
  mov [eax], edx

  mov edx, [edx]
  push IDispatch
  call dword [edx + Addref]

.ITypeInfo: mov eax, [ppTypeInfo]
  test eax, eax
  jz @F

  and dword [eax], 0

@@:  mov eax, S_OK
  return

proc IActiveScriptSite_GetDocVersionString, pi, pbstrVersionString
                enter
                mov eax, E_NOTIMPL
                return

proc IActiveScriptSite_OnScriptTerminate, pi, pvarResult, pexcepinfo
                enter
                mov eax, S_OK
                return

proc IActiveScriptSite_OnStateChange, pi, ssScriptState
                enter
                mov eax, S_OK
                return

proc IActiveScriptSite_OnScriptError, pi, pase
                enter
           invoke MessageBoxA, 0, mess1, mess1_title, MB_OK + MB_ICONERROR
  mov eax, S_OK
                return

proc IActiveScriptSite_OnEnterScript, pi
                enter
                mov eax, S_OK
                return

proc IActiveScriptSite_OnLeaveScript, pi
                enter
                mov eax, S_OK
                return

proc IActiveScriptSite_AddRef, pi
                enter
                mov ecx, [ActiveScriptSiteObject]
                inc dword [ecx + 12]
                mov eax, [ecx + 12]
                return

proc IActiveScriptSite_Release, pi
                enter
                mov ecx, [ActiveScriptSiteObject]
                dec dword [ecx + 12]
                mov eax, [ecx + 12]
                test eax, eax
                jnz @F
                invoke LocalFree, ecx
@@:         return

proc IActiveScriptSite_QueryInterface, pi, iid, ppvObject

                enter
             invoke IsEqualGUID, [iid], IID_IUnknown
                test eax, eax
                jnz .s_ok_unk
                invoke IsEqualGUID, [iid], IID_IActiveScriptSite
```

```
                test eax, eax
                jnz .s_ok_unk

                invoke IsEqualGUID, [iid], IID_IActiveScriptSiteWindow
                test eax, eax
                jnz .s_ok_acw

                invoke IsEqualGUID, [iid], IID_IDispatch
                test eax, eax
                jnz .s_ok_dsp

.NoInterface:   mov eax, E_NOINTERFACE
                return

.s_ok:  mov edx, [pIActiveScriptSite]
             mov edx, [edx]
             push [pIActiveScriptSite]
             call dword [edx + Addref]

  mov eax, S_OK
  return

.s_ok_unk:      mov eax, [ActiveScriptSiteObject]
                mov eax, [eax]
                mov edx, [ppvObject]
                mov [edx], eax
         jmp .s_ok

.s_ok_acw: mov eax, [ActiveScriptSiteObject]
                mov eax, [eax + 4]
                mov edx, [ppvObject]
                mov [edx], eax
      jmp .s_ok

.s_ok_dsp: mov eax, [ActiveScriptSiteObject]
                mov eax, [eax + 8]
                mov edx, [ppvObject]
                mov [edx], eax
      jmp .s_ok

proc IActiveScriptSiteWindow_GetWindow, pi, phwnd
  enter
  mov ecx, [phwnd]
      mov eax, [hWnd]
      mov [ecx], eax
      mov eax, S_OK
      return

proc IActiveScriptSiteWindow_EnableModeless, pi, fEnable
      enter
      mov eax, S_OK
      return

proc IDispatch_GetTypeInfoCount, pi, pctinfo
  enter
  mov eax, [pctinfo]
  mov dword [eax], 0
  mov eax, S_OK
  return

proc IDispatch_GetTypeInfo, pi, iTInfo, lcid, ppTInfo
  enter
  mov eax, DISP_E_BADINDEX
  return

proc IDispatch_GetIDsOfNames, pi, riid, rgszNames, cNames, lcid, rgDispId
  enter

  pushad

  mov esi, [rgszNames]
```

```
  mov ebx, [rgDispId]

  lodsd

  invoke lstrcmpW, eax, szSetText
  test eax, eax
  jnz @F

  mov dword [ebx], 1

  popad

  mov eax, S_OK
  return

@@:  or dword [ebx], -1
  popad

  mov eax, DISP_E_UNKNOWNNAME
  return

proc SetText, pNewText

  enter

      pushad
      mov ebx, [pNewText]

  invoke WideCharToMultiByte, CP_ACP, 0, ebx, -1, 0, 0, 0, 0

      push MEM_DECOMMIT
      push eax

      invoke VirtualAlloc, 0, eax, MEM_COMMIT, PAGE_READWRITE

      pop ecx
      push ecx
      push eax

      invoke WideCharToMultiByte, CP_ACP, 0, ebx, -1, eax, ecx, 0, 0

      invoke GetDlgItem, [hWnd], 1002
      mov ecx, [esp]
      invoke SetWindowTextA, eax, ecx

      invoke VirtualFree

      popad
      mov eax, S_OK
      return

proc IDispatch_Invoke, pi, dispIdMember, riid, lcid, wFlags, pDispParams, pVarResult, \
       pExcepInfo, puArgErr
  enter

  mov eax, [dispIdMember]
  dec eax
  jnz IDispatch_Invoke.notfound

  mov edx, [pDispParams]
  mov edx, [edx]
  mov edx, [edx + 8]

  push edx
   call SetText

  mov eax, S_OK
  return

.notfound: mov eax, DISP_E_MEMBERNOTFOUND
  return
```

```
proc GetFileName, hParent, lpTitle, lpFilter

  enter
  mov eax, [hParent]
      mov [ofn.hwndOwner], eax
     mov eax, [hInst]
     mov [ofn.hInstance], eax
     mov eax, [lpFilter]
      mov [ofn.lpstrFilter], eax
      mov eax, [szFileName]
      mov [ofn.lpstrFile], eax
      mov [ofn.nMaxFile], MAX_PATH
      mov eax, [lpTitle]
      mov [ofn.lpstrTitle], eax
      mov [ofn.Flags], OFN_EXPLORER or OFN_FILEMUSTEXIST or OFN_LONGNAMES

  invoke GetOpenFileName, ofn

  return

section '.data' data readable writeable

ofn OPENFILENAMEA

mess1   db "Cant't parse it!", 0
mess1_title  db 'script engine', 0

szFilter db 'VBScripts', 0, '*.vbs', 0, \
     'All files', 0, '*.*', 0, 0

szTitle  db 'Just open script-file...', 0
szFileName  dd 0

hInst dd 0

ActiveScriptSiteObject  dd 0
hWnd    dd 0
hFile    dd 0
FileSize   dd 0
buffer_size  dd 0
pMemory   dd 0
pScript   dd 0

Item_name dw "E", "d", "i", "t", "1", 0
szSetText dw "S", "e", "t", "T", "e", "x", "t", 0

pIActiveScript   dd 0
pIActiveScriptParse  dd 0
pIActiveScriptSite  dd 0

CLSID_VBScript GUID 0b54f3741h, 5b07h, 11cfh, 0a4h, 0b0h, 0, 0aah, 0, \
                    4ah, 55h, 0e8h
IID_IActiveScript GUID 0bb1a2ae1h, 0a4f9h, 11cfh, 8fh, 20h, 0, 80h, \
                       5fh, 2ch, 0d0h, 64h
IID_IActiveScriptParse GUID 0bb1a2ae2h, 0a4f9h, 11cfh, 8fh, 20h, 0, \
                            80h, 5fh, 2ch, 0d0h, 64h
IID_IActiveScriptSite GUID 0d57d7817h, 0e9b7h, 04a82h, 85h, 74h, 01h, \
                           0d0h, 0f9h, 3dh, 61h, 70h
IID_IActiveScriptSiteWindow GUID 0d10f6761h, 083e9h, 011cfh, 8fh, 20h, \
                                 0, 80h, 5fh, 2ch, 0d0h, 64h
IID_IUnknown GUID 0, 0, 0, 0ch, 0, 0, 0, 0, 0, 0, 46h
IID_IDispatch GUID 00020400h, 0, 0, 0c0h, 0, 0, 0, 0, 0, 0, 46h

IActiveScriptSite dd IActiveScriptSite_vtable
IActiveScriptSiteWindow dd IActiveScriptSiteWindow_vtable
IDispatch dd IDispatch_vtable

IActiveScriptSite_vtable dd IActiveScriptSite_QueryInterface
                         dd IActiveScriptSite_AddRef
                         dd IActiveScriptSite_Release
                         dd IActiveScriptSite_GetLCID
```

```
                        dd IActiveScriptSite_GetItemInfo
                        dd IActiveScriptSite_GetDocVersionString
                        dd IActiveScriptSite_OnScriptTerminate
                        dd IActiveScriptSite_OnStateChange
                        dd IActiveScriptSite_OnScriptError
                        dd IActiveScriptSite_OnEnterScript
                        dd IActiveScriptSite_OnLeaveScript

IActiveScriptSiteWindow_vtable dd IActiveScriptSite_QueryInterface
                        dd IActiveScriptSite_AddRef
                        dd IActiveScriptSite_Release
    dd IActiveScriptSiteWindow_GetWindow
    dd IActiveScriptSiteWindow_EnableModeless

IDispatch_vtable dd IActiveScriptSite_QueryInterface
  dd IActiveScriptSite_AddRef
  dd IActiveScriptSite_Release

  dd IDispatch_GetTypeInfoCount
  dd IDispatch_GetTypeInfo
  dd IDispatch_GetIDsOfNames
  dd IDispatch_Invoke

QueryInterface  = 0
Addref          = 04h
Release         = 08h
Close   = 1ch
SetScriptSite  = 0ch
InitNew  = 0ch
ParseScriptText = 14h
SetScriptState  = 14h
AddNamedItem = 20h

section '.idata' import data readable writeable
library kernel32, 'kernel32.dll', \
 user32, 'user32.dll', \
        ole32, 'ole32.dll', \
        comdlg32, 'comdlg32.dll'

import kernel32, ExitProcess, 'ExitProcess', \
     LocalAlloc, 'LocalAlloc', \
          LocalFree, 'LocalFree', \
          GetModuleHandleA, 'GetModuleHandleA', \
          CreateFileA, 'CreateFileA', \
          CloseHandle, 'CloseHandle', \
   ReadFile, 'ReadFile', \
          GetFileSize, 'GetFileSize', \
          MultiByteToWideChar, 'MultiByteToWideChar', \
          WideCharToMultiByte, 'WideCharToMultiByte', \
          VirtualAlloc, 'VirtualAlloc', \
          VirtualFree, 'VirtualFree', \
          lstrcmpW, 'lstrcmpW'

import ole32, CoInitialize, 'CoInitialize', \
              CoCreateInstance, 'CoCreateInstance', \
             IsEqualGUID, 'IsEqualGUID'

import user32, DialogBoxParamA, 'DialogBoxParamA', \
        EndDialog, 'EndDialog', \
        MessageBoxA, 'MessageBoxA', \
        SetWindowTextA, 'SetWindowTextA', \
        GetDlgItem, 'GetDlgItem'

import comdlg32, GetOpenFileName, 'GetOpenFileNameA'

section '.rsrc' resource data readable

directory RT_DIALOG, dialogs

resource dialogs, 1000, LANG_ENGLISH+SUBLANG_DEFAULT, main_dlg
dialog main_dlg, 'RunScript', 70, 70, 170, 70, \
```

```
       WS_CAPTION + WS_POPUP + WS_SYSMENU + DS_MODALFRAME

dialogitem 'BUTTON', 'Execute Script!', 1001, 52, 35, 55, 15, \
           WS_VISIBLE + WS_TABSTOP + BS_PUSHBUTTON
dialogitem 'EDIT', '', 1002, 10, 15, 150, 15, WS_VISIBLE + WS_BORDER + WS_TABSTOP

enddialog

;--------------------------------------------------------------------
; RunScript.inc
;--------------------------------------------------------------------

include 'include\win32ax.inc'

CLSCTX_INPROC_SERVER  equ 1
CLSCTX_INPROC_HANDLER  equ 2
CLSCTX_LOCAL_SERVER  equ 4
CLSCTX_INPROC_SERVER16  equ 8
CLSCTX_REMOTE_SERVER  equ 10h
CLSCTX_INPROC_HANDLER16  equ 20h
CLSCTX_INPROC_SERVERX86  equ 40h
CLSCTX_INPROC_HANDLERX86 equ 80h
CLSCTX_ESERVER_HANDLER  equ 100h

E_NOTIMPL  equ 80004001h
E_NOINTERFACE  equ 80004002h
E_INVALIDARG  equ 80070057h

TYPE_E_ELEMENTNOTFOUND      equ 8002802Bh

S_OK                         equ 0

LMEM_FIXED                  equ 0h

E_OUTOFMEMORY                equ 8007000Eh

SCRIPTSTATE_UNINITIALIZED  equ 0h
SCRIPTSTATE_STARTED        equ 1h
SCRIPTSTATE_CONNECTED        equ 2h
SCRIPTSTATE_DISCONNECTED   equ 3h
SCRIPTSTATE_CLOSED         equ 4h
SCRIPTSTATE_INITIALIZED    equ 5h

CP_ACP    equ 0

SCRIPTITEM_ISVISIBLE  equ 000000002h
SCRIPTITEM_ISSOURCE  equ 000000004h
SCRIPTITEM_GLOBALMEMBERS equ 000000008h
SCRIPTITEM_ISPERSISTENT  equ 000000040h
SCRIPTITEM_CODEONLY  equ 000000200h
SCRIPTITEM_NOCODE  equ 000000400h
SCRIPTITEM_ALL_FLAGS  equ 00000064eh

SCRIPTINFO_IUNKNOWN  equ 000000001h
SCRIPTINFO_ITYPEINFO  equ 000000002h
SCRIPTINFO_ALL_FLAGS  equ 000000003h

CC_STDCALL  equ     4h

DISPATCH_METHOD  equ 1
DISPATCH_PROPERTYGET equ 2
DISPATCH_PROPERTYPUT equ 4
DISPATCH_PROPERTYPUTREF equ 8
DISP_E_BADINDEX  equ  8002000Bh
DISP_E_MEMBERNOTFOUND equ  80020003h
DISP_E_UNKNOWNNAME equ  80020006h

VT_EMPTY            equ 0
VT_NULL             equ 1
VT_I2               equ 2
VT_I4               equ 3
```

```
VT_R4               equ 4
VT_R8               equ 5
VT_CY               equ 6
VT_DATE             equ 7
VT_BSTR             equ 8
VT_DISPATCH         equ 9
VT_ERROR            equ 10
VT_BOOL             equ 11
VT_VARIANT          equ 12
VT_UNKNOWN          equ 13
VT_DECIMAL          equ 14
VT_I1               equ 16
VT_UI1              equ 17
VT_UI2              equ 18
VT_UI4              equ 19
VT_I8               equ 20
VT_UI8              equ 21
VT_INT              equ 22
VT_UINT             equ 23
VT_VOID             equ 24
VT_HRESULT          equ 25
VT_PTR              equ 26
VT_SAFEARRAY        equ 27
VT_CARRAY           equ 28
VT_USERDEFINED      equ 29
VT_LPSTR            equ 30
VT_LPWSTR           equ 31
VT_RECORD           equ 36
VT_FILETIME         equ 64
VT_BLOB             equ 65
VT_STREAM           equ 66
VT_STORAGE          equ 67
VT_STREAMED_OBJECT  equ 68
VT_STORED_OBJECT    equ 69
VT_BLOB_OBJECT      equ 70
VT_CF               equ 71
VT_CLSID            equ 72
VT_BSTR_BLOB        equ 0fffh
VT_VECTOR           equ 1000h
VT_ARRAY            equ 2000h
VT_BYREF            equ 4000h
VT_RESERVED         equ 8000h
VT_ILLEGAL          equ 0ffffh
VT_ILLEGALMASKED    equ 0fffh
VT_TYPEMASK         equ 0fffh

struc GUID dd1, dw1, dw2, db1, db2, db3, db4, db5, db6, db7, db8
{
.dd1 dd dd1
.dw1 dw dw1
.dw2 dw dw2
.db1 db db1
.db2 db db2
.db3 db db3
.db4 db db4
.db5 db db5
.db6 db db6
.db7 db db7
.db8 db db8
}

MAX_PATH equ 260

struc OPENFILENAMEA
{
.lStructSize   dd 4ch
.hwndOwner          dd 0
.hInstance          dd 0
.lpstrFilter        dd 0
.lpstrCustomFilter  dd 0
.nMaxCustFilter     dd 0
```

```
  .nFilterIndex       dd 0
  .lpstrFile          dd 0
  .nMaxFile           dd 0
  .lpstrFileTitle     dd 0
  .nMaxFileTitle      dd 0
  .lpstrInitialDir    dd 0
  .lpstrTitle         dd 0
  .Flags              dd 0
  .nFileOffset        dw 0
  .nFileExtension     dw 0
  .lpstrDefExt        dd 0
  .lCustData          dd 0
  .lpfnHook           dd 0
  .lpTemplateName     dd 0
  }
  struct OPENFILENAMEA
```

Продолжение ожидается...

Файл к статье (находится в архиве `wasm_files.zip`: `files/0305/runscript2.zip`)

3 кита COM. Кит первый: реестр

Автор: Roustem

Дата: 2006-12-01

Скачать материалы для статьи (находится в архиве `wasm_files.zip`*:* `files/recovered/443/comkit1.rar`*)*

Эта статья была написана несколько лет назад. Может, она так и осталась бы валяться в архиве на CD среди прочего хлама, если бы я недавно не наткнулся на нее, разбирая старые материалы. Хотя задумывалось это в свое время как серия из трех статей, а в наличии пока только две (и я не знаю, сподвигнусь ли завершить эту серию), и даже несмотря на высказываемое некоторыми мнение, что "COM-де нынче устарел, а модно .NET", при перечитывании мне самому стало интересно, а вместе с тем и жалко, что такой материал пропадает - раз уж он был в свое время создан, я решил сделать его доступным для всех желающих, если такие найдутся.

Технология COM для многих программистов так и остается желанным, но слишком дорогим и поэтому недоступным ресурсом. Лишь беглый взгляд на все это нагромождение объектов, интерфейсов, правил их взаимодействия, отягощенный способами их реализации на том или ином языке и громоздкими вспомогательными конструкциями тех или иных сред разработки, призванными "облегчить" использование COM, способен надолго отбить охоту заниматься этим вопросом. А тот счастливчик, кому все же удалось продраться сквозь эти дебри и который вроде начал использовать объекты COM в своих программах, зачастую оказывается совершенно беспомощным вне своей привычной среды программирования.

В чем-то подобной этой ситуации оказался в свое время и автор этих строк, за тем разве исключением, что ему так и не удалось продраться через все эти услужливо придуманные высокоуровневые конструкции. Двигаясь вместо этого с противоположной стороны, я вдруг с удивлением обнаружил, что казавшаяся ранее неприступной преграда начинает отступать, если выкинуть изрядную долю отягощающего хлама. Проблема с COM не в том, что эта технология сложна сама по себе или что она использует какие-то неизвестные доселе принципы. Сами используемые элементы как раз просты и обыкновенны, но их много и они перемешаны со множеством других вещей, не имеющих отношения к COM, и все это приводит к тому, что суть COM теряется и растворяется в этой массе. В этой статье (вернее, серии из трех статей, в соответствии с названием) я попытаюсь описать мои собственные эксперименты и тот путь, который я прошел в надежде, что это может пригодиться кому-нибудь еще .

Предполагается, что читатель уже знаком или хотя бы слышал об основных концепциях COM. Здесь не будут описываться модели компонентного программирования, принципы инкапсуляции, полиморфизма, наследования и т.п. - разве что они будут упомянуты мимоходом. Вместо этого будет предложен подход с точки зрения низкоуровневого программирования, не отягощенного теоретическими абстракциями - своего рода "вид снизу" на архитектуру COM. Возможно, больше всего эти статьи подойдут тем, кто не раз начинал и бросал изучение COM, не справившись и изобилием разнородной информации, но все же не оставил желания разобраться в существе и механизмах этой технологии, а также тем, кто работает с COM на высоком уровне, но хотел бы заглянуть "за кулисы".

В качестве примера рассматриваются несколько утилит, созданных с использованием пакета MASM32. Предполагается, что читатель умеет работать с ним - вопросы создания графических Win32-приложений здесь вообще не будут рассматриваться. (Желающие могут обратиться к обучалкам Iczelion'a; в частности, хорошенько просмотрите раздел о диалоговых окнах - главы 10 и 11.) Более того, исходный код содержит некоторую мешанину - высокоуровневые конструкции

MASM (.IF-.ELSE, INVOKE и т.д.) используются для "обыкновенного" каркаса Win32-приложения, а элементы COM реализованы на самом низком уровне - например, в качестве виртуальных таблиц используются даже не структуры (не говоря уже о макросах), а ряд последовательных значений. Это сделано намеренно, чтобы акцентировать внимание именно на имеющих отношение к COM деталях и дать возможность "почувствовать" их и "пощупать руками", по возможности не зацикливаясь на Win32-программировании вообще.

COM и реестр

Ну что ж, пора покончить со вступлениями и перейти к нашим китам. Одна из важнейших основ, которая активно используется и без которой невозможна сама инфраструктура COM, - это системная база данных. На платформе Windows в качестве таковой выступает системный реестр. Прежде чем двинуться дальше, придется все же сказать пару слов предупреждения.

Здесь будут описаны эксперименты с изменениями значений в реестре. Следует помнить, что эти операции нужно производить очень аккуратно, поскольку они чреваты большими неприятностями. При порче реестра компьютер перестанет загружаться, и информация на диске может оказаться утраченной. Поэтому настоятельно рекомендуется перед началом манипуляций с реестром создать аварийную загрузочную дискету (если она не была создана до этого), сделать резервную копию всех важных данных на диске (не забыв и про электронную почту с адресной книгой), а также скопировать файлы реестра (для Win9* - USER.DAT и SYSTEM.DAT в каталоге Windows) в отдельную папку. Сложнее обстоит дело с WinNT/2k/XP, установленным в раздел NTFS. Для WinXP нужно хотя бы создать дополнительную точку восстановления.

Это лишь чрезвычайный минимум сведений; рассказ о методах восстановления реестра сам потребовал бы отдельной статьи. Существует серия книг с названиями "Реестр Windows [98, NT, 2000, XP и др.]", в которых говорится и о методах восстановления реестра; в случае необходимости следует обратиться именно к ним.

Вернемся к нашим китам. Давайте посмотрим на несколько небольших примеров. Если на вашей системе установлен MS Excel, вы можете создать такой файл с расширением .vbs и запустить его:

```
Set x = CreateObject("Excel.Application")
x.Workbooks.Add
x.Cells(1,1).Value = 5
x.Cells(1,2).Value = 10
x.Cells(1,3).Value = 15
x.Range("A1:C1").Select
Set ch = x.Charts.Add()
x.Visible = True
ch.Type = -4100
```

Если установлен Word XP, то можно запустить другой файл .vbs:

```
Set w = CreateObject("Word.Application")
w.Visible = True
Set rng = w.Document.Add.Range(0,0)
With rng
 .InsertBefore "Текст для WordXP"
 .ParagraphFormat.Alignment = 1
 With .Font
  .Name = "Arial"
  .Size = 16
  .Color = 200
 End With
End With
```

Если при установке MS Office был установлен элемент управления "Календарь", можно создать и просмотресть в браузере такой html-файл:

```
<HTML>
<BODY>
<OBJECT CLASSID="CLSID:8E27C92B-1264-101C-8A2F-040224009C02">
```

```
</OBJECT>
</BODY>
</HTML>
```

[Кстати, если этот элемент управления действительно установлен на вашем компьютере, вы можете увидеть его здесь под этим абзацем, если читаете статью в IE]

Вы, наверное, обратили внимание на постоянное повторение "если". Это неотъемлемое свойство компонентного ПО: к сожалению, никогда нельзя быть на 100% уверенным в том, что нужный компонент будет присутствовать на платформе пользователя. К тому же, простого копирования файлов для запусков компонентов недостаточно. Программы должны быть установлены либо с использованием setup, либо отдельной регистрацией компонентов или созданием соответствующих записей в реестре вручную.

После этого становится возможным обращение к компонентам по именам. В нашем случае это были соответственно "Excel.Application", "Word.Application" и "CLSID:8E27C92B-1264-101C-8A2F-040224009C02". Вся эта и другая относящаяся к COM/OLE/ActiveX информация хранится в разделе реестра "HKEY_LOCAL_MACHINE\Software\CLASSES". Этот раздел настолько важен, что для него был создан алиас под видом отдельного корневого раздела реестра "HKEY_CLASSES_ROOT".

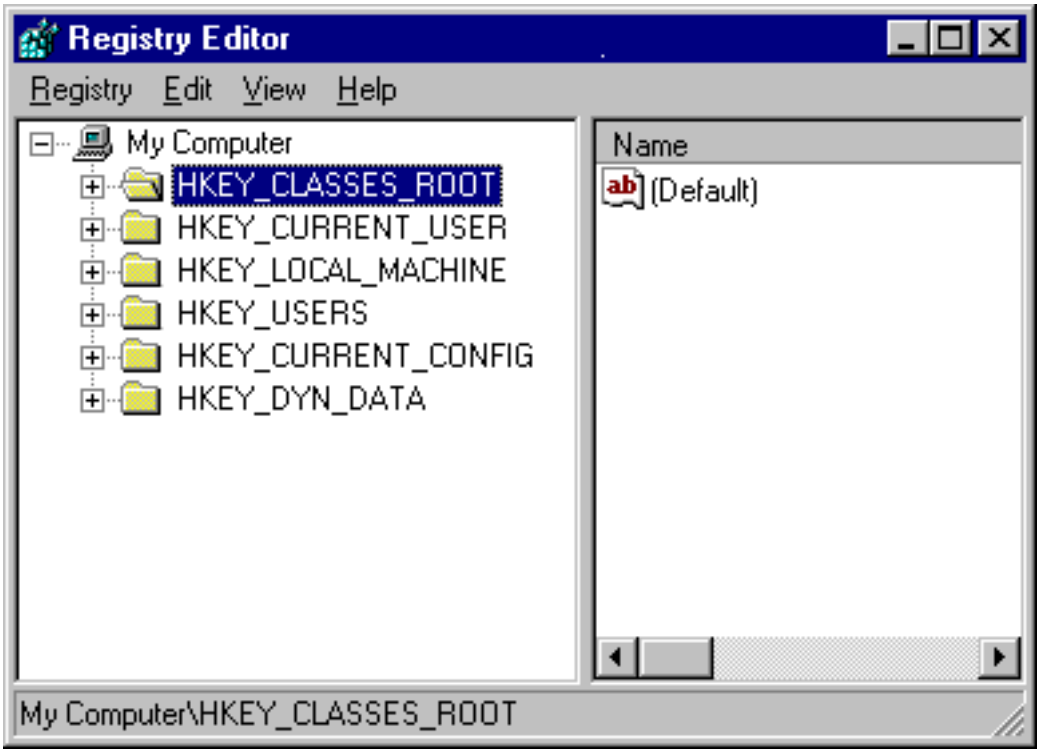


Рис. 1. Редактор реестра.

Структура информации о COM в реестре

Теперь самое время запустить редактор реестра (набрав в командной строке "regedit"; см. рис. 1) и начать знакомиться с рассматриваемыми вопросами на практике. Относящиеся к COM данные объединяются в 5 групп, для большинства из которых предусмотрены соответствующие разделы реестра: ProgID, CLSID, TypeLib, Interface, AppID. В качестве раздела для ProgID выступает сам корневой раздел "HKEY_CLASSES_ROOT". Вот здесь-то мы и можем обнаружить те самые имена "Excel.Application", "Word.Application", которые мы использовали ранее в скриптах, наряду с расширениями файлов и оставшимися 4 разделами для COM.

ProgID - это так называемый программный идентификатор, он является на самом деле лишь "дружественной для пользователя" меткой "настоящего" имени компонента - CLSID. Считается, что

ProgID не способен обеспечить подлинной глобальной уникальности имени, особенно в распределенной межсетевой среде, поэтому в последнем случае в качестве идентификаторов компонентов должны использоваться исключительно CLSID. Но на локальной машине, особенно в скриптах, ProgID широко используется. Однако в любом случае для идентификации компонента используется его CLSID; для этой цели между ключами CLSID и ProgID компонента установлены перекрестные ссылки.

Рассмотрим, например, ключ "Excel.Application". Он имеет подключ CLSID, значение по умолчанию которого и является искомым CLSID в строковом представлении. Это значит, что в разделе "HKEY_CLASSES_ROOT\CLSID" имеется соответствующий ключ, содержащий всю необходимую для загрузки компонента информацию. Наверняка вы обратили внимание и на то, что рядом с ключом "Excel.Application" имеется еще один такой же ключ, только с цифрой в конце (на разных машинах она может быть разной; у меня, например, "Excel.Application.5"); а сам ключ содержит также другой подключ - CurVer. На самом деле именно "Excel.Application.5" и является ProgID, а "Excel.Application" называется VersionIndependentProgID - "независимый от версии идентификатор прогаммы" (именно такие подключи для перекрестных ссылок вы найдете в разделе "HKEY_CLASSES_ROOT\CLSID" для соотвествующего компонента). Таким образом, в скрипте можно не указывать конкретную версию установленного компонента; а если со временем был сделан апгрейд, старые скрипты будут успешно работать с новыми версиями, если в качестве имен объектов они использовали VersionIndependentProgID. Для получения одного ключа из другого система COM предоставляет вспомогательные функции API CLSIDFromProgID и ProgIDFromCLSID.

Чтобы получше впитать в себя эту теорию, можно слегка похулиганить (не забывая, однако, сделанного ранее предупреждения). Убедимся, что "настоящее" имя компонента - это действительно CLSID. Для этого создадим новый ключ в разделе "HKEY_CLASSES_ROOT" и назовем его для разнообразия, скажем, собственным именем. Затем под этим ключем создаем подключ с именем "CLSID". Скопируем значение подключа CLSID для ProgID "Excel.Application" и вставим его в качестве значения по умолчанию нашего вновь созданного CLSID. А теперь слегка перепишем наш скрипт:

```
Set x = CreateObject("MyName")
x.Visible = True
```

Естественно, вместо "MyName" у вас должно быть то имя, которое вы выбрали для своего ProgID. Если все было сделано правильно, теперь Excel послушно "откликается" на новое имя.

Утилита для ProgID

Копаясь в реестре, можно обнаружить множество интересных вещей. Однако, ручной поиск по перекрестным ссылкам, особенно если компонентов очень много, весьма утомителен. Сочетая приятное с полезным, создадим для наших поисков небольшую утилиту, которая будет переводить данный ProgID компонента в его CLSID и наоборот.

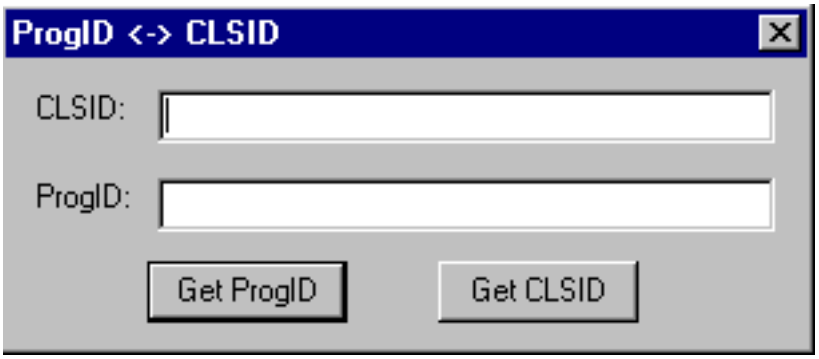


Рис.2. Интерфейс утилиты COM_1

Дизайн пользовательского интерфейса утилиты приведен на рис. 2. Он реализуется следующим файлом ресурсов (COM_1.rc):

```
#define DS_MODALFRAME       0x80L
#define DS_CENTER           0x0800L
#define WS_POPUP            0x80000000L
#define WS_CAPTION          0x00C00000L
#define WS_SYSMENU          0x00080000L
#define ES_AUTOHSCROLL      0x0080L

#define IDC_EDIT1           1000
#define IDC_EDIT2           1001
#define IDB_ProgID          1002
#define IDB_CLSID           1003
#define IDC_STATIC          -1

MyDialog DIALOG DISCARDABLE  200, 200, 200, 66
STYLE DS_MODALFRAME | DS_CENTER | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "ProgID <-> CLSID"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "Get ProgID",IDB_ProgID,35,46,50,14
    PUSHBUTTON      "Get CLSID",IDB_CLSID,108,46,50,14
    LTEXT           "CLSID:",IDC_STATIC,7,7,26,12
    LTEXT           "ProgID:",IDC_STATIC,7,27,26,12
    EDITTEXT        IDC_EDIT1,38,7,154,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT2,38,27,154,12,ES_AUTOHSCROLL
END
```

Исходный код программы содержится в файле COM_1.asm. Начало стандартное; функции COM API содержатся в OLE32.dll, поэтому необходимо включить файлы для поддержки этого модуля.

```
.386
.model flat,stdcall
option casemap:none

DlgProc proto :DWORD,:DWORD,:DWORD,:DWORD

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\ole32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\ole32.lib

.data

DlgName db "MyDialog",0
App  db "ProgID_CLSID",0
ms1  db "Enter CLSID here",0
ms2  db "Enter ProgID here",0
ms3  db "There is no ProgID for this CLSID",0
ms4  db "There is no CLSID for this ProgID",0
Err1  db "The wrong CLSID",0

.data?

hInstance HINSTANCE ?
hEdit1 DWORD ?  ; окно редактирования для CLSID
hEdit2 DWORD ?  ; окно редактирования для ProgID
buf  db 256 dup(?) ; для строк ANSI
wbuf  db 512 dup(?) ; для строк Unicode
cls  db 16 dup(?) ; буфер для CLSID
bufaddr DWORD ?

.const

IDC_EDIT1 equ 1000 ; окно редактирования для CLSID
IDC_EDIT2 equ 1001 ; окно редактирования для ProgID
IDB_ProgID equ 1002 ; кнопка "Get ProgID"
```

```
IDB_CLSID equ 1003 ; кнопка "Get CLSID"
```

Утилита реализована в виде модального диалогового окна, создаваемого из шаблона ресурса:

```
.code

start:
invoke GetModuleHandle, NULL
mov    hInstance,eax
invoke DialogBoxParam, hInstance, ADDR DlgName,NULL,
addr  DlgProc, NULL
invoke ExitProcess,eax

DlgProc proc USES esi edi ebx,hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
.IF uMsg==WM_INITDIALOG
 ; сохранить описатели для окон редактирования
 invoke GetDlgItem,hWnd,IDC_EDIT1
 mov hEdit1,eax
 invoke GetDlgItem,hWnd,IDC_EDIT2
 mov hEdit2,eax
 invoke SetFocus,hEdit1
 mov eax,FALSE
 ret
.ELSEIF uMsg==WM_CLOSE
 invoke EndDialog,hWnd,0
```

Алгоритм работы следующий. При нажатии кнопки "Get ProgID" окно для ProgID очищается и считывается текст окна для CLSID.

```
.ELSEIF uMsg==WM_COMMAND
 mov eax,wParam
 mov edx,wParam
 shr edx,16
 .if dx==BN_CLICKED
  .if ax==IDB_ProgID ; преобразовать CLSID в ProgID
   mov buf,0
   invoke SetWindowText,hEdit2,ADDR buf
   invoke GetWindowText,hEdit1,ADDR buf,255
```

Если текста нет - вывести сообщение с предложением его ввести:

```
.if eax==0 ; GetWindowText - нет текста
 invoke MessageBox,0,ADDR ms1,ADDR App,0
 invoke SendMessage,hEdit1,EM_SETSEL,0,-1
 invoke SetFocus,hEdit1
.else  ; текст получен
```

Здесь нас ожидает небольшой сюрприз - в COM используются строки в формате Unicode. Поэтому придется преобразовать текст в буфере buf с помощью функции MultiByteToWideChar. Эта функция принимает 6 параметров:

• кодовая страница преобразуемого текста (в нашем случае - ANSI, параметр CP_ACP);
• флаги, указывающие на способ преобразования сложных композитных символов; нам это не требуется - оставляем 0;
• адрес буфера с преобразуемой строкой;
• число символов в преобразуемой строке; если -1, строка заканчивается нулем и длина строки определяется автоматически;
• адрес буфера для преобразованной строки;
• размер буфера для преобразованной строки.

```
   invoke MultiByteToWideChar,CP_ACP,0,ADDR buf,-1,ADDR wbuf,510
```

Еще одно затруднение. Функция ProgIDFromCLSID принимает в качестве аргумента адрес структуры с числовым представлением CLSID, а не с его строковым представлением. Подробнее этот нюанс мы обсудим в разделе о CLSID, а сейчас используем для преобразования строкового представления в числовой еще одну вспомогательную функцию COM API - CLSIDFromString (где cls

- оставленный под структуру CLSID буфер размером 16 байт):

```
 invoke CLSIDFromString,ADDR wbuf,ADDR cls
 .if eax==NOERROR
  invoke ProgIDFromCLSID,ADDR cls,ADDR bufaddr
  .if eax==S_OK
```

Мы получили в bufaddr строковое представление ProgID в формате Unicode, теперь нужно преобразовать его обратно в ANSI с помощью функции WideCharToMultiByte ("напарницей" MultiByteToWideChar). Она принимает 8 аргументов:

• кодовая страница, в которую нужно преобразовать строку Unicode; в нашем случае - снова CP_ACP;
• флаги, указывающие способ обработки неотображаемых символов. Нам это не требуется, оставляем 0;
• адрес строки Unicode;
• число символов в строке Uinicode. Если -1, строка считается завершающейся нулями и ее длина определяется автоматически;
• адрес буфера для преобразованной строки;
• размер буфера для преобразованной строки;
• символ по умолчанию - нам не нужно, оставляем 0;
• флаги, указывающие на способ использования символа по умолчанию - 0.

```
  invoke WideCharToMultiByte,CP_ACP,0,bufaddr,-1,ADDR buf,255,0,0
  invoke SetWindowText,hEdit2,ADDR buf
 .else ; не удалось преобразовать CLSID в ProgID -
  ; выдать соответствующее сообщение
  invoke SetWindowText,hEdit2,ADDR ms3
 .endif
```

Дальше - блок обработки ошибки функции CLSIDFromString, что рассматривается как неправильный формат введенной строки CLSID:

```
 .else ; ошибка CLSIDFromString
  invoke MessageBox,0,ADDR Err1,ADDR App,MB_OK OR MB_ICONERROR
  invoke SendMessage,hEdit1,EM_SETSEL,0,-1
  invoke SetFocus,hEdit1
 .endif ;NOERROR - CLSIDFromString
.endif ;GetWindowText
```

Блок обработки нажатия на кнопку "Get ProgID" завершен. Сходным же образом обрабатывается нажатие кнопки "Get CLSID".

```
.elseif ax==IDB_CLSID ;преобразовать ProgID в строку CLSID
 mov buf,0
 invoke SetWindowText,hEdit1,ADDR buf
 invoke GetWindowText,hEdit2,ADDR buf,255
 .if eax==0 ; нет текста в окне - сообщение об ошибке
  invoke MessageBox,0,ADDR ms2,ADDR App,0
  invoke SendMessage,hEdit2,EM_SETSEL,0,-1
  invoke SetFocus,hEdit2
 .else  ; текст ProgID получен - преобразовать в Unicode
  invoke MultiByteToWideChar,CP_ACP,0,ADDR buf,-1,ADDR wbuf,510
   ; преобразовать ProgID в числовую форму CLSID
  invoke CLSIDFromProgID,ADDR wbuf,ADDR cls
  .if eax==S_OK ; CLSID получено,
     ; преобразовать в строковую форму
   invoke StringFromCLSID,ADDR cls,ADDR bufaddr
     ; преобразовать строку Unicode в ANSI
   invoke WideCharToMultiByte,CP_ACP,0,bufaddr,-1,
ADDR  buf,255,0,0
   invoke CoTaskMemFree,bufaddr ; освободить память,
; выделенную при вызове StringFromCLSID
   invoke SetWindowText,hEdit1,ADDR buf
  .else ; не удалось получить CLSID из ProgID
   invoke SetWindowText,hEdit1,ADDR ms4
```

```
    .endif ;S_OK (CLSIDFromProgID)
   .endif ;GetWindowText
  .endif ;ax==IDB_ProgID
```

На этом обработка сообщений диалогового окна заканчивается:

```
    .endif ;BN_CLICKED
   .ELSE
    mov eax,FALSE
    ret
   .ENDIF ;uMsg
         mov eax,TRUE
         ret
DlgProc endp

end start
```

Файлы с исходным кодом данной утилиты (COM_1.rc и COM_1.asm, а также makefile) находятся в каталоге COM_1 архива ComKit1.rar. Makefile написан в предположении, что пакет MASM32 установлен в каталоге \masm32 на текущем диске; если это не так, необходимо его отредактировать.

Раздел CLSID

Вся основная информация о компонентах размещается в разделе реестра "HKEY_CLASSES_ROOT\CLSID". Для каждого компонента создается отдельный ключ, в качестве имени которого выступает строковое представление его CLSID. Как отмечалось выше, CLSID (как и другие виды GUID, например, идентификаторы интерфейсов (IID) или библиотеки типов) имеет две формы представления: числовую и строковую. Числовая форма представляет собой 128-битное значение. В принципе это значение можно было бы разместить, скажем, в регистре XMM; однако COM создавалась еще во времена 16-разрядных машин, когда ни о каких XMM-регистрах и не помышляли. Поэтому для представления этого 128-битного значения была использована структура следующего вида:

```
GUID STRUCT
    Data1  dd ?
    Data2  dw ?
    Data3  dw ?
    Data4  db 8 dup(?)
GUID ENDS
```

В реестре же для представления GUID используется строка такого формата:

```
{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}
```

где x обозначает 16-ричную цифру (0-F), а фигурные скобки и дефисы должны присутствовать на своих местах без всяких разделяющих пробелов. На самом деле, эта запись может послужить источником путаницы: обратите внимание, что блок цифр для поля Data4 в строковой записи разделен на две части в 4 и 12 шестнадцатеричных цифр, и часть в 4 цифры можно спутать с предыдущими WORD-значениями. Первое DWORD и два последующих WORD значения записываются в строковом представлении, как обычные числа, т.е. старшие байты идут вначале, тогда как третий "WORD" (первые 2 байта поля Data4) записывается в порядке "little-endian" - вначале идет младший байт.

Как мы уже видели в коде первой утилиты, для преобразования строковой формы представления CLSID в числовую и обратно используются соотвтественно функции CLSIDFromString и StringFromCLSID. Существуют варианты и для других типов GUID - StringFromIID, IIDFromString, StringFromGUID2.

Ключ компонента CLSID содержит, в свою очередь, множество подключей. С двумя из них - ProgID и VersionIndependentProgID - мы уже познакомились. Стоит упомянуть лишь о том, что у компонента не обязательно должно присутствовать ProgID.

Простой универсальный клиент COM

Прежде, чем двинуться дальше в изучении раздела CLSID, нам понадобится еще одна небольшая утилита - универсальный клиент COM. Пользовательский интерфейс ее приведен на рис. 3. Этот клиент позволяет загружать объект COM, CLSID которого вводится в первое окно редактирования, и запрашивает у него указатель на интерфейс, IID которого содержится во втором окне редактирования. Тип сервера указывается набором флажков опций (одновременно может быть указано несколько типов). Кнопка IUnknown позволяет отобразить во втором окне редактирования IID часто употребляемого интерфейса с тем же названием.

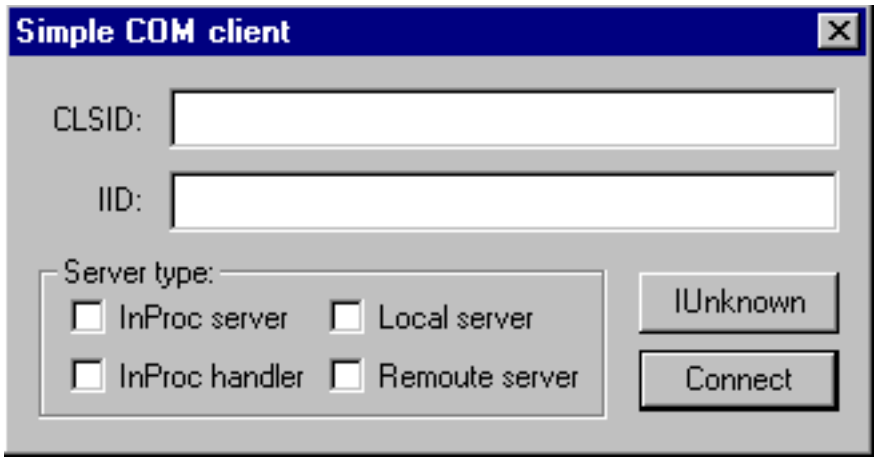


Рис.3. Интерфейс утилиты COM_2

Вот соответствующий файл ресурсов (COM_2.rc):

```
#include "\masm32\include\resource.h"

#define IDC_EDIT1  1001
#define IDC_EDIT2  1002
#define IDC_CHECK1 1003
#define IDC_CHECK2 1004
#define IDC_CHECK3 1005
#define IDC_CHECK4 1006
#define IDC_BUTTON1 1007
#define IDC_STATIC -1

MyDialog DIALOGEX 0, 0, 214, 90
STYLE DS_MODALFRAME | DS_CENTER | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Simple COM client"
FONT 8, "MS Sans Serif"
BEGIN
    LTEXT           "CLSID:",IDC_STATIC,7,10,26,8,0,WS_EX_RIGHT
    LTEXT           "IID:",IDC_STATIC,7,28,26,8,0,WS_EX_RIGHT
    GROUPBOX        "Server type:",IDC_STATIC,7,45,142,38
    EDITTEXT        IDC_EDIT1,40,7,167,14,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT2,40,26,167,14,ES_AUTOHSCROLL
    CONTROL         "InProc server",IDC_CHECK1,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,15,55,58,10
    CONTROL         "InProc handler",IDC_CHECK2,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,15,68,62,10
    CONTROL         "Local server",IDC_CHECK3,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,80,55,55,10
    CONTROL         "Remoute server",IDC_CHECK4,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,80,68,66,10
    PUSHBUTTON      "IUnknown",IDC_BUTTON1,157,49,50,14
    DEFPUSHBUTTON   "Connect",IDOK,157,67,50,14
END
```

Чтобы не выписывать каждый раз стандартные define'ы, целесообразно скопировать файл "resource.h" из 10 урока обучалки Iczelion'a в каталог "Include" пакета MASM32, а затем вставлять в rc-файлы соответствующую ссылку, как это сделано в данном случае.

Реализация в файле COM_2.asm:

```
.386
.model flat,stdcall
option casemap:none

DlgProc proto :DWORD,:DWORD,:DWORD,:DWORD

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\ole32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\ole32.lib

.data

DlgName db "MyDialog",0
app  db "SimpleClient",0
ms1  db "Enter CLSID here",0
ms2  db "Enter IID here",0
szUnk  db "{00000000-0000-0000-C000-000000000046}",0
mscheck db "The server type must be indicated",0
msclsid db "Enter CLSID here",0
msiid  db "Enter IID here",0
msok db "Got interface pointer to:",13,10,"CLSID:",9,"%s",13,10,"IID:",9,"%s",0
mserr  db "CoCreateInstance failed:",13,10,"HRESULT==%Xh",0
err1  db "The wrong CLSID",0
err2  db "The wrong IID",0

.data?

hInstance HINSTANCE ?
hEdit1 DWORD ?  ; "CLSID"
hEdit2 DWORD ?  ; "IID"
hCheck1 DWORD ?  ; "Inproc server"
hCheck2 DWORD ?  ; "Inproc handler"
hCheck3 DWORD ?  ; "Local server"
hCheck4 DWORD ?  ; "Remoute server"
ctx  DWORD ?  ; тип сервера
szclsid db 64 dup(?) ; буфер для строки с CLSID
sziid  db 64 dup(?) ; буфер для строки с IID
buf  db 256 dup(?) ; для ANSI строк
wbuf  db 512 dup(?) ; для строк Unicode
cls  db 16 dup(?) ; CLSID компонента
iid  db 16 dup(?) ; IID вызываемого интерфейса
pUnk  DWORD ?  ; указатель на интерфейс IUnknown

.const

IDC_EDIT1 equ 1001 ; "CLSID"
IDC_EDIT2 equ 1002 ; "IID"
IDC_CHECK1 equ 1003 ; "Inproc server"
IDC_CHECK2 equ 1004 ; "Inproc handler"
IDC_CHECK3 equ 1005 ; "Local server"
IDC_CHECK4 equ 1006 ; "Remoute server"
IDC_BUTTON1 equ 1007 ; "IUnknown"

.code

start:
 invoke GetModuleHandle, NULL
 mov    hInstance,eax
 invoke DialogBoxParam, hInstance, ADDR DlgName,NULL, addr DlgProc, NULL
 invoke ExitProcess,eax

DlgProc proc USES esi edi ebx,hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
.IF uMsg==WM_INITDIALOG
 ; сохраняем описатели элементов управления
 invoke GetDlgItem,hWnd,IDC_EDIT1
```

```
 mov hEdit1,eax
 invoke GetDlgItem,hWnd,IDC_EDIT2
 mov hEdit2,eax
 invoke GetDlgItem,hWnd,IDC_CHECK1
 mov hCheck1,eax
 invoke GetDlgItem,hWnd,IDC_CHECK2
 mov hCheck2,eax
 invoke GetDlgItem,hWnd,IDC_CHECK3
 mov hCheck3,eax
 invoke GetDlgItem,hWnd,IDC_CHECK4
 mov hCheck4,eax
 invoke SetFocus,hEdit1
```

Поскольку на этот раз нам необходимо задействовать библиотеку COM, нужно также вызвать функцию CoInitialize. При завершении программы необходимо соответственно вызвать функцию CoUninitialize.

```
 invoke CoInitialize,0
 mov eax,FALSE
 ret
.ELSEIF uMsg==WM_CLOSE
 invoke CoUninitialize
 invoke EndDialog,hWnd,0
```

"Ядром" данной утилиты является использование функции CoCreateInstance, которая делает всю работу - локализует соответствующий объект COM, загружает его, запрашивает указанный интерфейс и возвращает указатель на него. Значительная часть кода обработки сообщения WM_COMMAND сводится к сбору данных для передачи в качестве параметров функции CoCreateInstance и проверке введенных значений. CoCreateInstance принимает 5 аргументов:

• адрес структуры, содержащей CLSID запрашиваемого объекта (мы получаем это значение в первом окне редактирования);
• указатель на так называемый внешний IUnknown, если объект агрегируется в состав другого объекта. Если агрегирование не применяется (как в нашем случае), значение аргумента равно 0;
• контекст создания объекта: флаги, указывающие на то, является ли сервер объекта внутрипроцессным, локальным, удаленным или это внутрипроцессный обработчик (данный флаг мы вычисляем на основе проверки состояния флажков опций);
• адрес структуры с IID интерфейса, указатель на который мы хотим получить. Значение IID мы получаем во втором окне редактирования;
• адрес переменной, в которой будет возвращен указатель на запрашиваемый нами интерфейс (или 0, если этот интерфейс не поддерживается).

При нажатии на кнопку IUnknown во второе окно редактирования копируется IID для соответствующего интерфейса (это значение "вшито" в код в виде строки szUnk). Итак, обработка команд:

```
.ELSEIF uMsg==WM_COMMAND
 mov eax,wParam
 mov edx,wParam
 shr edx,16
 .if dx==BN_CLICKED
  .if ax==IDC_BUTTON1 ; кнопка 'IUnknown'
invoke SetWindowText,hEdit2,offset szUnk
  .elseif ax==IDOK  ; кнопка 'Connect'
; получить CLSID
invoke GetWindowText,hEdit1,offset szclsid,64
.if eax==0 ; текст отсутствует - сообщение
 invoke MessageBox,0,offset msclsid,offset app,MB_ICONEXCLAMATION
 invoke SendMessage,hEdit1,EM_SETSEL,0,-1
 invoke SetFocus,hEdit1
 mov eax,TRUE
 ret
.else  ; текст получен - перевести в ANSI и
  ; преобразовать в числовую форму, сохранив в cls
```

```
invoke MultiByteToWideChar,CP_ACP,0,offset szclsid,-1,offset wbuf,510
 invoke CLSIDFromString,offset wbuf,offset cls
 .if eax!=NOERROR ; ошибка CLSIDFromString рассматривается
    ; как неверный формат введенной строки с CLSID
  invoke MessageBox,0,offset err1,offset app,MB_ICONERROR
  invoke SendMessage,hEdit1,EM_SETSEL,0,-1
  invoke SetFocus,hEdit1
  mov eax,TRUE
  ret
 .endif ;NOERROR - CLSIDFromString
.endif ; GetWindowText (hEdit1)

; получить IID
invoke GetWindowText,hEdit2,offset sziid,64
.if eax==0 ; текста нет - сообщение
 invoke MessageBox,0,offset msiid,offset app,MB_ICONEXCLAMATION
 invoke SendMessage,hEdit2,EM_SETSEL,0,-1
 invoke SetFocus,hEdit2
 mov eax,TRUE
 ret
.else ; текст с IID получен, преобразовать в ANSI,
 ; затем в числовую форму, сохранив ее в iid
 invoke MultiByteToWideChar,CP_ACP,0,offset sziid,-1,offset wbuf,510
 invoke IIDFromString,offset wbuf,offset iid
 .if eax!=NOERROR ; ошибка IIDFromString рассматривается
    ; как неверный формат введенной строки с IID
  invoke MessageBox,0,offset err2,offset app,MB_ICONERROR
  invoke SendMessage,hEdit2,EM_SETSEL,0,-1
  invoke SetFocus,hEdit2
  mov eax,TRUE
  ret
 .endif ;NOERROR - IIDFromString
.endif ;GetWindowText (hEdit2)

; проверить состояние флажков опций и определить тип сервера,
; сохранив полученное значение в ctx
mov ctx,0
invoke IsDlgButtonChecked,hWnd,IDC_CHECK1
.if eax==BST_CHECKED
 or ctx,CLSCTX_INPROC_SERVER
.endif
invoke IsDlgButtonChecked,hWnd,IDC_CHECK2
.if eax==BST_CHECKED
 or ctx,CLSCTX_INPROC_HANDLER
.endif
invoke IsDlgButtonChecked,hWnd,IDC_CHECK3
.if eax==BST_CHECKED
 or ctx,CLSCTX_LOCAL_SERVER
.endif
invoke IsDlgButtonChecked,hWnd,IDC_CHECK4
.if eax==BST_CHECKED
 or ctx,CLSCTX_REMOTE_SERVER
.endif
.if ctx==0 ; не выбран ни один тип сервера - сообщение
 invoke MessageBox,0,offset mscheck,offset app,MB_ICONEXCLAMATION
 invoke SetFocus,hCheck1
 mov eax,TRUE
 ret
.endif
```

Теперь необходимые параметры собраны: CLSID находится в cls, IID - в iid, тип сервера - в ctx. Функция CoCreateInstance находит объект COM с данным CLSID и создает его экземпляр, запрашивая у него указатель на интерфейс IID. Если объект реализует интерфейс с данным IID, он помещает указатель на него в переменную, адрес которой был передан CoCreateInstance в последнем параметре, а сама функция возвращает S_OK. Если объект не поддерживает данный интерфейс или объект с данным CLSID не зарегистрирован в реестре, или зарегистрированный тип сервера не указан в флаге типа сервера, возвращается сообщение об ошибке.

```
invoke CoCreateInstance,offset cls,0,ctx,offset iid,offset pUnk
```

```
mov ecx,eax
.if eax==S_OK ; Объект успешно создан и указатель на затребованный
   ; интерфейс возвращен в pUnk. Выводим сообщение с
   ; указанием CLSID объекта и IID интерфейса
 invoke wsprintf,offset buf,offset msok,offset szclsid,offset sziid
 invoke MessageBox,0,offset buf,offset app,MB_ICONINFORMATION
```

На этом вся "полезная" работа нашего приложения завершается - мы освобождаем объект. Делать это придется уже средствами COM, т.е. с использованием полученного указателя интерфейса. Для этого через полученный указатель интерфейса (pUnk) необходимо вызвать метод Release интерфейса IUnknown (абсолютно все интерфейсы COM наследуются от IUnknown, поэтому метод Release можно вызвать для любого объекта COM одним и тем же способом).

Указатель на интерфейс объекта представляет собой адрес некоей структуры, в которой содержатся данные с состоянием экземпляра объекта. Доступ к ним возможен лишь косвенно, посредством вызова методов соответствующего интерфейса, поэтому каждый метод получает в качестве первого параметра адрес этой структуры (т.н. указатель "this", в терминологии языка C++). Об этой структуре можно сказать лишь то, что первое ее поле является указателем на другую структуру - "виртуальную таблицу", содержащую адреса процедур, реализующих методы интерфейса. Для вызова того или иного метода необходимо найти адрес его реализации по заранее известному и постоянному смещению в виртуальной таблице. Подробнее "кухню" этого вопроса мы разберем во второй статье, а пока просто приведем код:

```
 mov eax,pUnk
 push eax  ; указатель на экземпляр объекта ("this")
 mov eax,[eax] ; получаем указатель на виртуальную таблицу,
 call dword ptr [eax+8] ; затем - указатель на третью функцию
; в таблице (Release) и вызываем ее

.else ; ошибка CoCreateInstance
 invoke wsprintf,offset buf,offset mserr,ecx
 invoke MessageBox,0,offset buf,offset app,MB_ICONERROR
.endif ; eax==S_OK (CoCreateInstance)
```

На этом обработка WM_COMMAND и любых других сообщений закончена.

```
    .endif ;ax==IDOK

   .endif ;BN_CLICKED

  .ELSE ; uMsg не обрабатывается
   mov eax,FALSE
   ret
  .ENDIF ; uMsg
        mov eax,TRUE
        ret
DlgProc endp

end start
```

Подключи раздела CLSID

Теперь можно продолжить эксперименты с реестром. Важнейшими подключами раздела "HKEY_CLASSES_ROOT\CLSID" является ряд подключей, позволяющих определить местонахождение сервера в системе: InprocServer, InprocServer32, InprocHandler, InprocHandler32, LocalServer, LocalServer32. Каждый тип сервера представлен двумя подключами; суффикс "32" означает, что сервер 32-разрядный, подключ без этого суффикса указывает на то, что сервер 16-разрядный (для совместимости со старыми компонентами). Случай удаленного (remoute) сервера обрабатывается особым образом; об этом мы поговорим в разделе, посвященном AppID.

Начнем хулиганить. При создании новых интерфейсов и компонентов для получения уникальных GUID используются специальные утилиты типа guidgen и т.п. Microsoft зарезервировала для собственного употребления большой интервал удобных GUID-ов с наборами нулей; мы же

поступим еще круче - используем GUID'ы типа {00000000-0000-0000-0000-000000000001}. Зачем нам париться, выискивая свой компонент среди массы цифр, когда можно удобно разместить его в самом начале раздела? Можно было бы (и вначале автор так и поступал) использовать и GUID со всеми нулями, но это специальный зарезервированный GUID_NULL, употребляющийся в особых случаях; во избежание неприятностей лучше оставить его в покое.

Создадим в разделе "HKEY_CLASSES_ROOT\CLSID" ключ {00000000-0000-0000-0000-000000000001}. Этот ключ может иметь значение по умолчанию, которое является описанием компонента, предназначенным для пользователя. Обратите внимание, это не то же самое, что ProgID (хотя некоторые авторы компонентов используют в качестве описания ProgID). Можно вставить сюда какую-нибудь строку, например, "Мой любимый компонент". Далее, создаем подключ с именем "LocalServer32". А вот значение по умолчанию этого подключа как раз и содержит полный путь к файлу сервера. Запишем сюда "c:\windows\calc.exe" (нужно указать каталог windows на вашей машине). Теперь запустим наш клиент (COM_2.exe), наберем CLSID, нажмем на IUnknown, в качестве сервера выбираем "Local server", жмем "Connect" и... Калькулятор можно закрыть. А наш клиент, похоже, завис - но к этому надо было быть готовым, поскольку калькулятор понятия не имеет о том, что он вдруг стал компонентом COM; тогда как система COM все еще дожидается от него ответа - наш клиент в это время "сидит внутри" CoCreateInstance. Впрочем, примерно через минуту она вернет ошибку таймаута, и наш клиент выдаст соответствующее окно сообщения.

Я не буду описывать здесь все прочие эксперименты - путь указан, утилиты есть; экспериментируйте! Технология COM постигается лишь на практике. Только не забудьте в пылу увлечения предупреждение об аккуратности работы с реестром.

Рассмотрим другие подключи. Важными являются TreatAs и AutoTreatAs, поскольку они позволяют сменить сервер компонента. Если в ключе с данным CLSID имеется подключ TreatAs (значение по умолчанию которого содержит другой CLSID), то будет создан объект с этим другим CLSID, независимо от наличия подключей InprocServer, LocalServer и т.д. Проделаем такой эксперимент. Создадим подключ "TreatAs" в нашем ключе {00000000-0000-0000-0000-000000000001}. В качестве значения вставим CLSID для компонента Excel.Application (воспользовавшись для преобразования ProgID утилитой COM_1.exe). Если теперь с помощью COM_2.exe попытаться загрузить наш компонент, будет загружен Excel (правда, он останется скрытым. Чтобы убедиться, что он запущен, придется посмотреть в менеджере задач).

Подключ TreatAs позволяет осуществлять эмуляцию одного сервера другим. Функция CoGetTreatAsClass позволяет получить значение этого ключа, а функция CoTreatAsClass - установить (или удалить) его. Если подключ TreatAs удаляется с помощью функции CoTreatAsClass, эта функция проверяет значение подключа AutoTreatAs. Если ключ с таким именем существует, его значение копируется в подключ TreatAs; если нет, подключ TreatAs вообще удаляется. Подключ же AutoTreatAs можно создавать или удалять только вручную, с использованием функций API реестра. Таким образом, в подключе AutoTreatAs может сохраняться значение предыдущего CLSID, если данный сервер эмулируется сначала одним, а затем другим сервером, обеспечивая своего рода "постоянную" эмуляцию (в отличие от "временной" эмуляции с помощью подключа TreatAs).

Будучи основным источником сведений о компоненте, раздел "HKEY_CLASSES_ROOT\CLSID" содержит подключи с перекрестными ссылками на другие разделы. О подключах ProgID и VersionIndependentProgID мы уже говорили. Подключ TypeLib содержит GUID, который является идентификатором библиотеки типов для данного компонента; соответствующий подключ содержится в разделе "HKEY_CLASSES_ROOT\TypeLib". Подключ AppID содержит GUID, идентифицирующий подключ другого раздела - "HKEY_CLASSES_ROOT\AppID", и указывает на то, что объект является распределенным (DCOM). Подробнее об этих подключах будет сказано в соответствующих разделах.

В разделе CLSID имеется еще множество других подключей; мы не будем рассматривать их все. Упомянем лишь о тех, которые являются своего рода флагами, указывающими на тип компонента. Наличие подключа "Control" указывает, что компонент является элементом управления; "Insertable" - что объект может внедряться в контейнеры OLE; "OLEScript" - объект является исполнителем сценариев (scripting engine); "DocObject" - объект-документ; "Printable" - объект может быть распечатан; "Programmable" - объект является сервером автоматизации (и, соответственно, им можно управлять с помощью скриптов); "Ole1Class" - объект OLE 1.0 (старой версии).

Утилита для просмотра типов объектов

Создадим еще одну небольшую утилиту, которая будет отображать список зарегистрированных в реестре объектов, относящихся к одному из 7 перечисленных в конце предыдущего параграфа типов. Внешний вид приложения показан на рис. 4. Тип объекта можно выбрать в выпадающем списке; после нажатия кнопки "List" в окне списка отображаются описания соответствующих объектов (значения по умолчанию, связанные с тем или иным CLSID). Если описания нет, строка останется пустой.

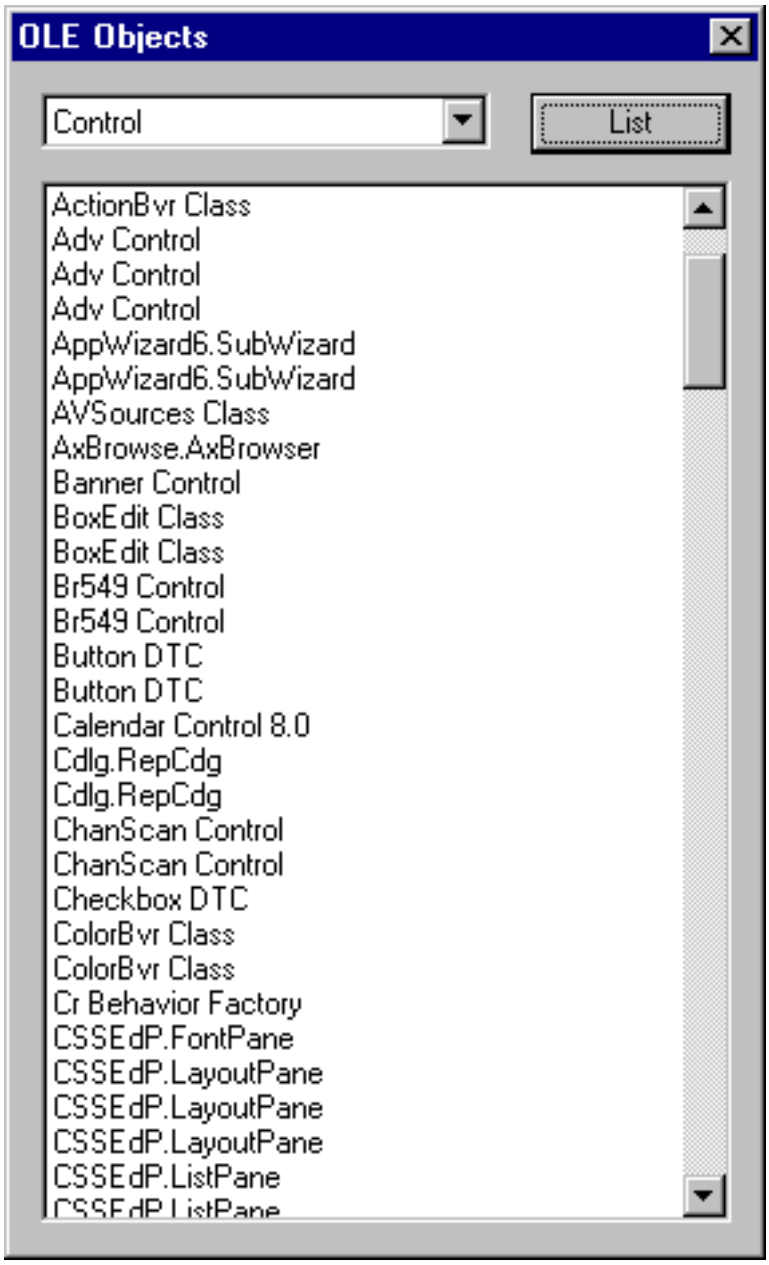


Рис.4. Интерфейс утилиты COM_3

Для экономии места мы не будем приводить здесь полный код приложения; соответствующие файлы находятся в каталоге COM_3 архива ComKit1.rar. Вместо этого рассмотрим лишь некоторые существенные моменты, имеющие отношение к логике работы программы.

```
invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey1,0,KEY_ALL_ACCESS,ADDR HKey
.if eax==ERROR_SUCCESS
```

Начинаем с того, что открываем раздел реестра "HKEY_CLASSES_ROOT\CLSID", сохраняя описатель в переменной HKey. При успешном открытии начинается основной цикл по перечислению имеющихся в открытом разделе ключей.

```
EnumLoop:
 mov sbksz,255
 invoke RegEnumKeyEx,HKey,idx,ADDR SubKeyBuf,ADDR sbksz,0,0,0,0
 .if eax==ERROR_NO_MORE_ITEMS
  jmp OutLoop
 .elseif eax!=ERROR_SUCCESS
  invoke MessageBox,0,ADDR Err2,ADDR App,MB_OK OR MB_ICONERROR
  jmp OutLoop
 .endif
```

При любой ошибке перечисления цикл прерывается, причем если ошибка не связана с просмотром всех ключей, выводится соответствующее сообщение. Для каждого нового найденного ключа составляются две строки вида: "CLSID\{clsid компонента}" и "CLSID\{clsid компонента}\<тип объекта>", где <тип объекта> означает одну из 7 строк с типом объекта, выбранную из выпадающего списка.

```
 invoke lstrcpy,ADDR SubKey2,ADDR SubKey1
 invoke lstrcpy,ADDR SubKey3,ADDR SubKey1
 invoke lstrcat,ADDR SubKey2,ADDR s
 invoke lstrcat,ADDR SubKey3,ADDR s
 invoke lstrcat,ADDR SubKey2,ADDR SubKeyBuf
 invoke lstrcat,ADDR SubKey3,ADDR SubKeyBuf
 invoke lstrcat,ADDR SubKey3,ADDR s
 invoke lstrcat,ADDR SubKey3,ADDR findbuf
 invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey3,0,
KEY_ALL_ACCESS,ADDR HKey2
.if eax==ERROR_SUCCESS
```

Затем делается попытка открыть ключ реестра "CLSID\{clsid компонента}\<тип объекта>". Успешная попытка означает, что ключ с данным CLSID имеет подключ с выбранным нами именем; в этом случае открываем другой подготовленный ключ ("CLSID\{clsid компонента}") и запрашиваем значение по умолчанию, которое пересылаем в окно списка и переходим к новой итерации:

```
 invoke RegCloseKey,HKey2
 invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey2,0,
KEY_ALL_ACCESS,ADDR HKey3
 invoke RegQueryValueEx,HKey3,0,0,0,ADDR namebuf,ADDR bsz
 invoke RegCloseKey,HKey3
 invoke SendMessage,hLst,LB_ADDSTRING,0,ADDR namebuf
.endif ;RegOpenKeyEx (2)
 inc idx
 jmp EnumLoop
OutLoop:
 invoke RegCloseKey,HKey
```

Раздел Interface

В отличие от остальных разделов, "HKEY_CLASSES_ROOT\Interface" относится не к отдельным компонентам, а к системе в целом. Приведенные здесь данные об интерфейсах используются при стандартном маршалинге. Подключей немного: BaseInterface, NumMethods, ProxyStubCLSID и ProxyStubCLSID32. BaseInterface содержит имя интерфейса, от которого унаследован данный интерфейс; если он не указан, в качестве базового принимается IUnknown. NumMethods содержит число методов в интерфейсе. ProxyStubCLSID(32) подобен InprocServer(32): он содержит полный

путь к серверу (dll), реализующему вспомогательные объекты (т.н. представители и заглушки), которые используются системой при маршалинге методов данного интерфейса. Как обычно, суффикс "32" в имени означает 32-разрядный сервер, его отсутствие - 16-разрядный.

Подробное обсуждение маршалинга и связанных с ним проблем выходит за рамки данной статьи; кое-что будет рассказано во второй статье, а здесь приведем описание полезной утилиты, использующей данный раздел реестра. Внешний вид ее приведен на рис. 5. При нажатии кнопки "List" приложение создает экземпляр COM-объекта, CLSID которого указан в окне редактирования. Затем у созданного объекта запрашиваются все возможные интерфейсы, сведения о которых имеются в системном реестре. В окне списка выводятся имена всех интерфейсов, которые реализованы объектом.

Рис.5. Интерфейс утилиты COM_4

Здесь опять приведем лишь ключевые моменты кода, исходные файлы содержатся в каталоге COM_4 архива ComKit1.rar.

Поскольку приложение использует библиотеку COM, при инициализации диалогового окна вызывается процедура CoInitialize, а при его закрытии - соответственно CoUninitialize. При нажатии

кнопки "List" окно списка очищается, а из окна редактирования извлекается CLSID уже знакомым нам способом:

```
invoke GetWindowText,hEd,ADDR buffer,255
invoke MultiByteToWideChar,CP_ACP,0,ADDR buffer,511,ADDR wbuf,255
invoke CLSIDFromString,ADDR wbuf,ADDR cls
```

Затем создается экземпляр объекта с данным CLSID; в качестве начального интерфейса запрашивается IUnknown (его IID содержится в IUnk), а контекст объекта допускает любой тип сервера:

```
invoke CoCreateInstance,ADDR cls,0, CLSCTX_SERVER,ADDR IUnk,ADDR pUnk
```

Если указатель успешно получен (в pUnk), открываем раздел реестра "HKEY_CLASSES_ROOT\Interface" и входим в главный цикл перечисления подключей (т.е. интерфейсов):

```
 invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey1,0,
KEY_ALL_ACCESS,ADDR HKey
 .if eax==ERROR_SUCCESS
Enum1:
 mov sbksz,255
 invoke RegEnumKeyEx,HKey,idx,ADDR SubKeyBuf,ADDR sbksz,0,0,0,0
 .if eax==ERROR_NO_MORE_ITEMS
  jmp Ex_2
 .elseif eax!=ERROR_SUCCESS
  invoke MessageBox,0,ADDR Err2,ADDR App,MB_OK OR MB_ICONERROR
  jmp Ex_2
```

Выход из цикла осуществляется, когда функция перечисления возвращает ошибку; причем если это не связано с отсутствием ключей для дальнейшего перечисления, отображается сообщение об ошибке. Каждый новый ключ перечисления (IID интерфейса в строковом виде) преобразуется в числовой вид (если в ходе преобразования происходит ошибка, данный ключ просто игнорируется и цикл продолжается).

```
.else
 invoke MultiByteToWideChar,CP_ACP,0,ADDR SubKeyBuf,255,ADDR wbuf,255
 invoke CLSIDFromString,ADDR wbuf,ADDR cls
 .if eax!=NOERROR
  jmp Cont1
 .endif
```

А вот теперь нужно запросить у нашего объекта интерфейс, IID которого мы только что получили. Для этого необходимо вызвать метод QueryInterface интерфейса IUnknown, со следующими аргументами:

• указатель на экземпляр текущего объекта ("this");
• адрес структуры с IID запрашиваемого интерфейса;
• адрес переменной, в которой будет возвращен указатель затребованного интерфейса.

Адрес метода QueryInterface располагается в самом начале виртуальной таблицы (со смещением 0):

```
lea eax,pItfc ; pItfc получит указатель нового интерфейса
push eax
lea eax,cls  ; требуемый IID находится в cls
push eax
mov eax,pUnk ; указатель на интерфейс IUnknown
   ; нашего объекта ("this")
push eax
mov eax,[eax] ; получаем адрес виртуальной таблицы
call dword ptr [eax] ; вызываем первую функцию
; из виртуальной таблицы (QueryInterface)
```

В случае успешного получения указателя нового интерфейса QueryInterface возвращает S_OK. Нам нужен просто факт поддержки объектом данного интерфейса, он сам не нужен; поэтому мы сразу

же освобождаем только что полученный указатель, вызвав через него метод Release:

```
.if eax==S_OK
 mov eax,pItfc ; указатель на затребованный интерфейс
 push eax  ; помещаем в качестве 1-го (и единственного) пар-ра;
 mov eax,[eax] ; адрес виртуальной таблицы
 call dword ptr [eax+8] ; вызываем 3-й метод из в.табл. (Release)
```

Одновременно с этим создаем строку вида "Interface\{IID интерфейса}" и открываем соответствующий раздел реестра, запрашивая значение по умолчанию этого раздела (имя интерфейса). Это имя добавляется в окно списка.

```
 invoke lstrcpy,ADDR SubKey2,ADDR SubKey1
 invoke lstrcat,ADDR SubKey2,ADDR s
 invoke lstrcat,ADDR SubKey2,ADDR SubKeyBuf
 invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey2,0,
KEY_ALL_ACCESS,ADDR HKey2
 .if eax==ERROR_SUCCESS
  mov bsz,255
  invoke RegQueryValueEx,HKey2,0,0,0,ADDR namebuf,ADDR bsz
  .if eax!=ERROR_SUCCESS
   invoke MessageBox,0,ADDR Err5,ADDR App,MB_OK OR MB_ICONERROR
  .endif
  invoke RegCloseKey,HKey2
  invoke SendMessage,hLst,LB_ADDSTRING,0,ADDR namebuf
 .endif ;RegOpenKeyEx (2)
.endif ;S_OK (QueryIntervace)
```

В случае, когда QueryInterface или RegOpenKeyEx возвращают код ошибки, данный ключ просто игнорируется, а цикл продолжается дальше.

```
Cont1:
 inc idx
 jmp Enum1
 .endif ;RegEnumKeyEx
Ex_2:
 invoke RegCloseKey,HKey
```

После выхода из цикла метод Release вызывается также через указатель pUnk интерфейса IUnknown, полученный нами при создании экземпляра объекта (аналогично тому, как это было сделано ранее).

Раздел TypeLib

В разделе "HKEY_CLASSES_ROOT\TypeLib" содержатся ключи, имена которых представляют собой строковую форму идентификаторов (GUID) установленных в системе библиотек типов. На данные ключи ссылаются подключи TypeLib компонентов раздела "HKEY_CLASSES_ROOT\CLSID", но обратных ссылок нет. Библиотеки типов обязательны для компонентов, использующих позднее связывание и автоматизацию; в частности, они широко применяются с элементами управления ActiveX. Остальные компоненты могут и не иметь библиотеки типов.

Каждый ключ раздела TypeLib содержит иерархическую структуру подключей. На первом уровне находятся подключи версии библиотеки типов, представленные в строковой форме major.minor (цифры, соответствующие версии). В разделе версии находятся, в свою очередь, подключи HelpDir (полный путь к файлу справки), Flags (флаг библиотеки типов) и строковое представление локального идентификатора языка (lcid). Ключ <lcid> содержит еще один или два подключа: win16 и/или win32 - полные пути к библиотеке типов для соответствующей платформы.

В качестве примера работы с библиотекой типов рассмотрим еще одно приложение (рис. 6). Диалоговое окно приложения имеет всего один элемент управления - окно просмотра списков. В первом столбце отображаются описания компонентов, а рядом с ними во втором - идентификаторы соответствующих библиотек типов. При двойном щелчке мышью на названии компонента либо отображается общая информация о данном компоненте, содержащаяся в его библиотеке типов,

либо (если имеется) открывается файл справки, связанный с данным компонентом.

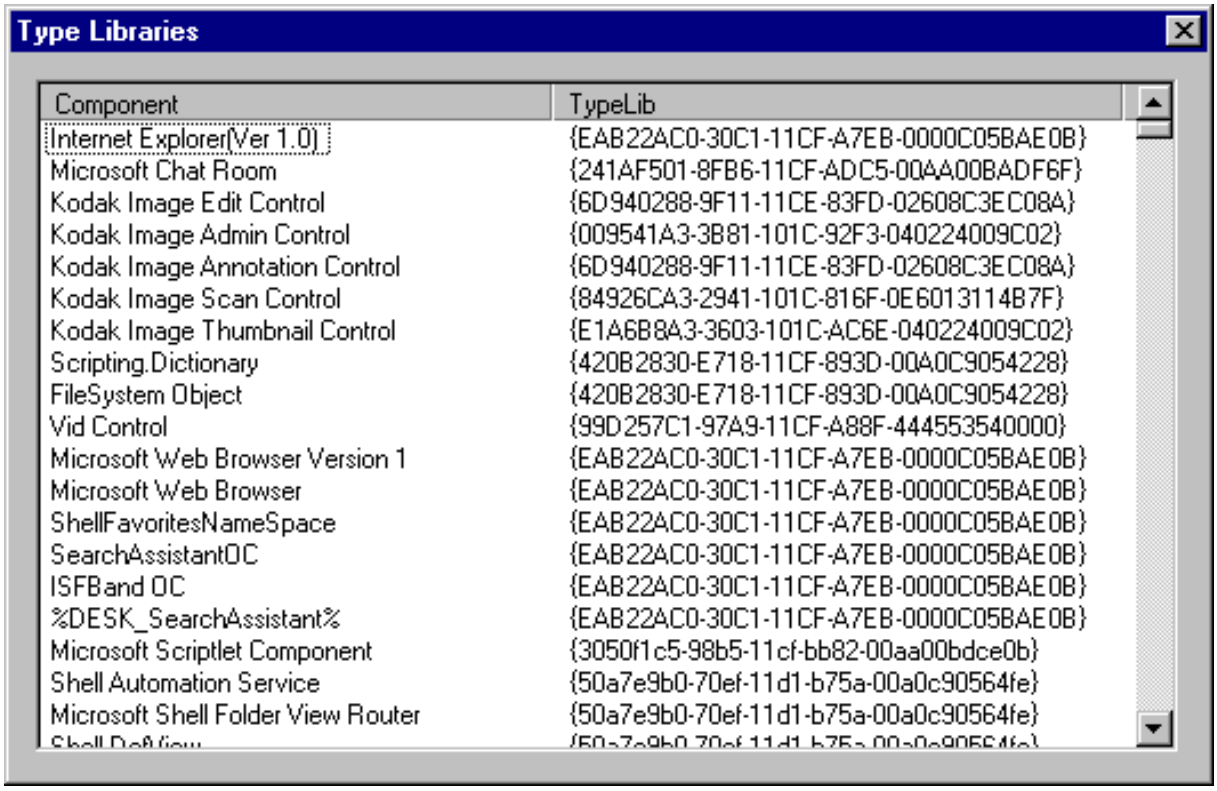


Рис.6. Интерфейс утилиты COM_5

Приложение работает следующим образом (детали реализации графического интерфейса опущены; полный исходный код содержится в каталоге COM_5 архива ComKit1.rar). При обработке сообщения WM_INITDIALOG открываем раздел реестра "HKEY_CLASSES_ROOT\CLSID", затем входим в главный цикл перечисления содержащихся в нем ключей:

```
invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey1,0,KEY_ALL_ACCESS,ADDR HKey
 .if eax==ERROR_SUCCESS
EnumLoop:
  lea esi,buf
  mov sbksz,255
  invoke RegEnumKeyEx,HKey,idx,ADDR SubKeyBuf,ADDR sbksz,0,0,0,0
  .if eax==ERROR_NO_MORE_ITEMS
   jmp OutLoop
  .elseif eax!=ERROR_SUCCESS
   invoke MessageBox,0,ADDR Err2,ADDR App,MB_OK OR MB_ICONERROR
   jmp OutLoop
  .endif
```

Как и в предыдущей утилите, если RegEnumKeyEx возвращает ошибку, цикл прерывается. Для каждого нового ключа составляются по две строки вида: "CLSID\{clsid компонента}" и "CLSID\{clsid компонента}\TypeLib", затем делается попытка открыть раздел реестра со второй строкой в качестве имени.

```
 invoke lstrcpy,ADDR SubKey2,ADDR SubKey1
 invoke lstrcpy,ADDR SubKey3,ADDR SubKey1
 invoke lstrcat,ADDR SubKey2,ADDR s
 invoke lstrcat,ADDR SubKey3,ADDR s
 invoke lstrcat,ADDR SubKey2,ADDR SubKeyBuf
 invoke lstrcat,ADDR SubKey3,ADDR SubKeyBuf
 invoke lstrcat,ADDR SubKey3,ADDR s
 invoke lstrcat,ADDR SubKey3,ADDR findbuf
 invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey3,0,
KEY_ALL_ACCESS,ADDR HKey3
```

Успешное открытие ключа TypeLib говорит о том, что для компонента имеется библиотека типов. В этом случае запрашиваем значение ключа "CLSID\{clsid компонента}" и помещаем соответствующую строку в первую колонку окна просмотра списков, а значение ключа "CLSID\{clsid компонента}\TypeLib" (строковое представление идентификатора библиотеки типов) - во вторую колонку:

```
.if eax==ERROR_SUCCESS
invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,
ADDR SubKey2,0,KEY_ALL_ACCESS,ADDR HKey2
 .if eax==ERROR_SUCCESS
mov bsz,255
invoke RegQueryValueEx,HKey2,0,0,0,ADDR buf,ADDR bsz
  mov eax,cnt
mov item.iItem,eax
mov item.iSubItem,0
invoke SendMessage,hList,LVM_INSERTITEM,0,ADDR item
 mov bsz,255
 invoke RegQueryValueEx,HKey3,0,0,0,ADDR buf,ADDR bsz
  .if eax==ERROR_SUCCESS
   mov item.iSubItem,1
   invoke SendMessage,hList,LVM_SETITEM,0,ADDR item
  .endif
  invoke RegCloseKey,HKey2
  inc cnt
 .endif ;RegOpenKeyEx (SubKey2)
 invoke RegCloseKey,HKey3
.endif ;RegOpenKeyEx (SubKey3)
inc idx
jmp EnumLoop
OutLoop:
 invoke RegCloseKey,HKey
```

Двойной щелчок на элементе окна просмотра списка обрабатывается в сообщении WM_NOTIFY. Соответствующий код вынесен в отдельную процедуру FindHelp, причем в регистре ebx передается адрес структуры NMLISTVIEW с информацией об элементе списка, на котором был произведен двойной щелчок:

```
.ELSEIF uMsg==WM_NOTIFY
 mov ebx,lParam
 assume ebx:ptr NMHDR
 .if [ebx].code==NM_DBLCLK
  call FindHelp
 .endif
 assume ebx:nothing
```

Процедура FindHelp начинается с проверки того, что щелчок действительно произведен на элементе списка:

```
FindHelp proc
 assume ebx:ptr NMLISTVIEW
 mov eax,[ebx].iItem
 mov item.iItem,eax
 .if eax==-1
  ret
 .endif
 assume ebx:nothing
```

Теперь можно скопировать в буфер buf содержимое второй колонки соответствующего элемента (GUID библиотеки типов в текстовом виде).

```
 mov item.imask,LVIF_TEXT
 mov item.iSubItem,1
 mov item.pszText,OFFSET buf
 mov item.cchTextMax,255
 invoke SendMessage,hList,LVM_GETITEM,0,ADDR item
```

Из полученного значения составляем название раздела реестра "TypeLib\{GUID библиотеки}" и открываем его.

```
  invoke lstrcpy,ADDR SubKey2,ADDR cl2
  invoke lstrcat,ADDR SubKey2,ADDR s
  invoke lstrcat,ADDR SubKey2,ADDR buf
  invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey2,
0,KEY_ALL_ACCESS,ADDR HKey
```

Нужно обработать не очень удобную форму представления данных для библиотеки типов. Чтобы получить путь к файлу библиотеки типов, необходимо сначала указать версию, а о ней заранее невозможно сказать ничего определенного. Плохо то, что функция для загрузки LoadRegTypeLib требует точного указания старшей (major) версии и отказывается загружать библиотеки с другими версиями. Поэтому придется "вручную" просмотреть подключи с версиями и выделить из имени подключа старшую версию.

```
  .if eax==ERROR_SUCCESS
   mov idx,0
   mov mjver,0 ; для нахождения max значения majversion
TlibLoop:
   mov sbksz,255
   ; перечислим подключи в разделе 'HKCR\TypeLib\{GUID}'
   ; (в форме majversion.minversion)
   invoke RegEnumKeyEx,HKey,idx,ADDR SubKeyBuf,ADDR sbksz,0,0,0,0
   .if eax==ERROR_SUCCESS
    xor eax,eax
    mov al,SubKeyBuf ; первый символ подключа (т.е. majversion)
    sub eax,30h  ; символ преобразуем в значение
    cmp mjver,eax
    jge Tlib1  ; выбираем для версии большее значение
    mov mjver,eax
Tlib1:
    inc idx
    jmp TlibLoop
   .endif ;RegEnumKeyEx
   invoke RegCloseKey,HKey
```

Мы получили номер старшей версии; теперь можно обычным способом преобразовать строковое представление GUID в числовое и передать всё функции LoadRegTypeLib, которая принимает следующие аргументы:

• адрес структуры с GUID загружаемой библиотеки;
• старшая версия загружаемой библиотеки (необходимо указать точно; в нашем случае она находится в переменной mjver);
• младшая версия загружаемой библиотеки (здесь можно оставить 0, потому что система загрузит любую младшую версию, превышающую указанную);
• локальный языковой идентификатор (lcid; можно оставить нейтральный - 0);
• адрес переменной, в которой будет возвращен указатель на интерфейс ITypeLib.

```
  invoke MultiByteToWideChar,CP_ACP,0,ADDR buf,-1,ADDR wbuf,510
  invoke CLSIDFromString,ADDR wbuf,ADDR TlibGuid
  .if eax==NOERROR
   invoke LoadRegTypeLib,ADDR TlibGuid,mjver,0,0,ADDR pTlib
   .if eax==S_OK
```

Возвращенное значение S_OK говорит о том, что у нас есть действительный указатель на интерфейс ITypeLib. Нас интересует метод GetDocumentation, имеющий следующие аргументы:

• индекс описания типа; если -1, возвращаются данные для самой библиотеки типов (что нам и нужно);
• адрес переменной, в которой будет возвращен указатель на строку типа BSTR с именем запрашиваемого элемента (в нашем случае – самой библиотеки типов);

• адрес переменной, в которой будет возвращен указатель на строку типа BSTR с описанием запрашиваемого элемента;
• адрес переменной, в которой будет возвращен контекстный идентификатор для файла справки;
• адрес переменной, в которой будет возвращен указатель на строку типа BSTR с полным путем к файлу справки.

Если какой-нибудь вид данных не нужен, можно передать в соответствующем аргументе 0. Адрес функции GetDocumentation расположен в виртуальной таблице интерфейса ITypeLib по смещению 24h.

```
push OFFSET HelpFile
push 0  ; контекст не нужен, pBstrHelpCtx=NULL
push OFFSET DocString
push OFFSET CompName
push -1  ; индекс (-1='сама библиотека')
mov eax,pTlib
push eax  ; указатель ‘this’
mov eax,[eax] ; виртуальная таблица
call dword ptr [eax+24h] ;GetDocumentation
```

Если по возвращении из метода в переменной HelpFile окажется 0, это значит, что файла справки нет. В этом случае выведем простое окно сообщения с полученными сведениями:

```
.if HelpFile==0
 invoke WideCharToMultiByte,CP_ACP,0,CompName,-1,ADDR SubKeyBuf,255,0,0
 invoke lstrcpy,ADDR wbuf,ADDR SubKeyBuf
 invoke lstrcat,ADDR wbuf,ADDR CRLF
 invoke WideCharToMultiByte,CP_ACP,0,DocString,-1,ADDR SubKeyBuf,255,0,0
 invoke lstrcat,ADDR wbuf,ADDR SubKeyBuf
 invoke lstrcat,ADDR wbuf,ADDR CRLF
 invoke lstrcat,ADDR wbuf,ADDR ms1
 invoke MessageBox,0,ADDR wbuf,ADDR App,MB_OK or MB_ICONINFORMATION
```

Если же файл справки указан, преобразуем соответствующую строку из Unicode в ANSI и передадим ее адрес в качестве аргумента функции ShellExecute (чтобы иметь возможность загружать как hlp-, так и chm-файлы):

```
.else   ; загрузить и отобразить файл справки
 invoke WideCharToMultiByte,CP_ACP,0,HelpFile,-1,ADDR buf,255,0,0
 invoke ShellExecute,0,ADDR cmd,ADDR buf,0,0,SW_SHOW
 .if eax<33 ; ошибка ShellExecute
  invoke wsprintf,ADDR wbuf,ADDR fmt,ADDR buf
  invoke MessageBox,0,ADDR wbuf,ADDR App,MB_OK or MB_ICONERROR
 .endif
.endif  ;HelpFile==0
```

Строки типа BSTR необходимо освободить, во избежание утечки ресурсов:

```
invoke SysFreeString,HelpFile
invoke SysFreeString,CompName
invoke SysFreeString,DocString
```

При ошибке вызова метода GetDocumentation отображаем соответствующее сообщение. В любом случае после этого необходимо освободить указатель интерфейса ITypeLib (pTlib):

```
.else    ; ошибка вызова GetDocumentation
 invoke MessageBox,0,ADDr Err6,ADDR App,MB_OK or MB_ICONERROR
.endif   ; GetDocumentation
mov eax,pTlib
push eax   ; указатель ‘this’
mov eax,[eax]  ; виртуальная таблица
call dword ptr [eax+8] ; функция №3 (Release)
```

Завершают функцию обработки других ошибок:

```
 .else  ; ошибка LoadRegTypeLib
  invoke MessageBox,0,ADDR Err5,ADDR App,MB_OK or MB_ICONERROR
 .endif ; LoadRegTypeLib==S_OK
```

```
  .else   ; ошибка CLSIDFromString
   invoke MessageBox,0,ADDR Err4,ADDR App,MB_OK or MB_ICONERROR
  .endif  ; CLSIDFromString==NOERROR
 .else    ; ошибка RegOpenKeyEx
  invoke MessageBox,0,ADDR Err7,ADDR App,MB_OK or MB_ICONERROR
  ret
 .endif   ; RegOpenKeyEx==ERROR_SUCCESS
 ret
 FindHelp endp
```

С помощью данной утилиты можно обнаружить множество интересных вещей. Попробуйте!

Раздел AppID

Раздел реестра «HKEY_CLASSES_ROOT\AppID» совместно с разделом «HKEY_LOCAL_MACHINE\Software\Microsoft\OLE» определяют установки для распределённой системы COM (DCOM).

Распределённая система имеет дело с передачей данных по сети, и перед ней сразу же встают вопросы аутентификации, авторизации и другие проблемы обеспечения безопасности. Общая схема конфигурирования обычно такова: в разделе «HKEY_LOCAL_MACHINE\Software\Microsoft\OLE» описываются значения параметров безопасности по умолчанию. Раздел «HKEY_CLASSES_ROOT\AppID» содержит ключи (GUID), в которых параметры безопасности могут задаваться для отдельных групп компонентов (в этом случае «HKEY_CLASSES_ROOT\CLSID\{clsid компонента}\AppID» содержит ссылку на соответствующий ключ раздела «HKEY_CLASSES_ROOT\AppID»). Значения из раздела AppID имеют преимущество перед соответствующими значениями по умолчанию. Кроме того, параметры безопасности и другие связанные с распределенной системой параметры могут задаваться явным образом при создании компонента с использованием функции CoCreateInstanceEx или ей подобной. В этом случае используются явно заданные значения.

Раздел «HKEY_LOCAL_MACHINE\Software\Microsoft\OLE» может иметь следующие именованные значения:

• EnableDCOM – разрешает (Y или y) или запрещает (N или n) удаленным клиентам запуск компонентов на данной системе;
• DefaultLaunchPermission – определяет список участников (ACL), кто может запускать по умолчанию компоненты на данной машине;
• DefaultAccessPermission – определяет список участников (ACL), кто может по умолчанию получать доступ к запущенным объектам;
• LegacyAuthenticationLevel – устанавливает уровень аутентификации по умолчанию;
• LegacyImpersonationLevel – устанавливает уровень имперсонации по умолчанию;
• LegacyMutualAuthentication – разрешает (Y или y) или запрещает (любое другое значение или отсутствие именованного значения) взаимную аутентификацию;
• LegacySecureReferences – указывает, охраняются (Y или y) или нет (любое другое значение или отсутствие именованного значения) вызовы AddRef и Release.

Ключи «HKEY_CLASSES_ROOT\AppID\{GUID}» могут иметь следующие именованные значения:

• RemoteServerName – имя сервера, на котором должен быть запущен компонент;
• ActivateAtStorage – указывает, что компонент должен быть запущен на той же машине, на которой хранятся данные объекта;
• LocalService – указывает, что компонент реализован в виде Win32-сервиса;
• ServiceParameters – используется совместно с LocalService. Значение этого параметра передается при запуске соответствующего сервиса в виде аргументов командной строки;
• RunAs – позволяет запустить компонент, не являющийся сервисом, от имени определенного пользователя;

• LaunchPermission – определяет список участников (ACL), кто может запустить сервер с данным компонентом;
• AccessPermission – определяет список участников (ACL), кто может получить доступ к объектам данного класса;
• DllSurrogate – содержит полный путь к программе-оболочке (exe), в которой должен быть запущен удаленный внутрипроцессный (dll) сервер;
• AuthenticationLevel – определяет уровень аутентификации для компонента.

Указанные в реестре значения могут быть перекрыты явным вызовом функции CoInitializeSecurity.

3 whales of COM. The first whale: registry

Автор: Roustem

Дата: 2006-12-26

Download materials for the article (находится в архиве `wasm_files.zip`*:* `files/recovered/444/comkit1.rar`*)*

This article was written a few years ago. Maybe it would still lie among other stuff on a CD if I didn't find it recently examining old material. Though I intended to write a series of three articles in due time and now there are only two ones (I don't know will it be completed ever), and despite of some peoples' opinion that "COM is out of date now, up-to-date fancy is .NET ", it seemed interesting to me myself when I had reread it, and I felt sorry that such a thing was getting lost. So I decided to make it available to everybody interested since it had been created already.

COM technology remains desired but too expencive and therefore inaccessible resource for many programmers. Just fleeting glance on all that conglomeration of objects, interfaces, rules of their interaction overburdened with implementation techniques in this or that programming language and cumbersome auxiliary constructions intended for "facilitation" of COM using can kill any desire to engage in this matter. And that lucky beggar who succeeded in making his or her way through all this maze and who began as though to use COM objects in his or her own programs often finds himself or herself quite helpless outside of his or her usual programming environment.

The author of this article turned out to be in somewhat similar situation in his time except that he just could not pass through all these helpfully made-up high-level constructions. Moving instead from the opposite side I discovered suddenly that seeming earlier inaccessible barrier begun to step back if one threw away a fairly good part of this overburdening trash. The problem with COM is that not the technology is difficult in itself or that it uses some unknown so far principles. Elements used are just ordinary and simple but there is a great number of them and they are mixed with a lot of other things not related to COM; all this hides and diffuses the essence of COM. In this article (or rather in series of three articles in compliance with the title) I'll try to describe my own experiments and the way I have passed in hope it will be helpful for somebody else.

It is presumed that the reader is familiar with the basic concepts of COM or at least has heard about them. You won't find here descriptions of component programming models, principles of encapsulation, polymorphism, inheritance etc. - unless of only incidentally mentioning. Instead you'll be presented with the low-level programming approach without excess theoretical abstractions - the "bottom view" onto COM architecture in a way. Maybe these articles will best fit to those who repeatedly began and gave up studying COM beeng not able to cope with abundance of heterogeneous information, but who kept desire to understand the essence and machinery of this technology, ant to those who worked with COM on high level but who wanted to look "under hood".

Several utilities created using MASM32 package are examined as examples. It is supposed that the reader is able to work with this package - issues of creating Win32 GUI applications won't be considered here. (Those who wish that can address to Iczelion's tutorials; specifically you must look through the sections about dialog boxes - 10 and 11. Moreover the source code in this article contains some mixture of high-level MASM constructions (.IF-.ELSE, INVOKE etc.) used for "ordinary" skeletons of Win32 applications and of quite low-level implementation of COM elements - for example, not even structures (not to mention macros) but a set of successive values is used in place of virtual tables. It is done intentionally to attract attention just to details relating to COM and to give a chance to "feel" them without drawing away to Win32 programming in general.

COM and registry

So, it's time to give up with preambles and pass on to our whales. One of the most important actively used bases without which COM infrastructure is impossible at all is a system database. The system registry plays this role on Windows platform. Before we advance further it's necessary to say a couple of words of warning.

Experiments with modification of registry values will be described here. You must remember to be utterly cautious making such operations because you may fall in troubles. If the registry becomes corrupted the computer will cease from booting and information on disks may turn out to be lost. Therefore it is strongly recommended to create emergency repair disk (if it wasn't created earlier) before manipulations with the registry, make a backup copy of all important data on hard disks (including email and an address book) as well as backup registry files (for Win9* these are USER.DAT and SYSTEM.DAT in the Windows folder) in a separate folder. It's more complicated for WinNT/2k/XP installed on a NTFS partition. For WinXP it's necessary to create at least one additional recovery point.

It's only the exraordinary minimum of information; the description of registry repair methods would require a separate article in itself. There are series of books devoted to Windows [98, NT, 2000, XP etc.] registries; these are the books that you should address to when necessary.

Coming back to our whales. Let's consider a few small examples. If you have installed MS Excel on your system you can create the following file with the .vbs extension and launch it:

```
Set x = CreateObject("Excel.Application")
x.Workbooks.Add
x.Cells(1,1).Value = 5
x.Cells(1,2).Value = 10
x.Cells(1,3).Value = 15
x.Range("A1:C1").Select
Set ch = x.Charts.Add()
x.Visible = True
ch.Type = -4100
```

If you have installed Word XP you can launch another .vbs file:

```
Set w = CreateObject("Word.Application")
w.Visible = True
Set rng = w.Document.Add.Range(0,0)
With rng
 .InsertBefore "Text for WordXP"
 .ParagraphFormat.Alignment = 1
 With .Font
  .Name = "Arial"
  .Size = 16
  .Color = 200
 End With
End With
```

If the control "Calendar" was installed when installing MS Office you can create and examine in your browser the following html-file:

```
<HTML>
<BODY>
<OBJECT CLASSID="CLSID:8E27C92B-1264-101C-8A2F-040224009C02">
</OBJECT>
</BODY>
</HTML>
```

[By the way, if this control is indeed installed on your computer you can see it here below if you are reading the article in IE]

You probably have noticed the continual repetition of this "if". It's an integral feature of the component software: unfortunately, you can't be sure for 100% that the needed component will be present on a user's platform. Besides it's insufficient simply to copy files for loading components. The programs must be installed either by using setup or by separate registering of components or creating the necessary records in a registry by hand.

After that you can address components by their names. In our case these are "Excel.Application", "Word.Application", and "CLSID:8E27C92B-1264-101C-8A2F-040224009C02". All this and other COM/OLE/ActiveX related information is stored in registry under "HKEY_LOCAL_MACHINE\Software\CLASSES" key. This key is so important that an alias was created for it as the separate root key "HKEY_CLASSES_ROOT".

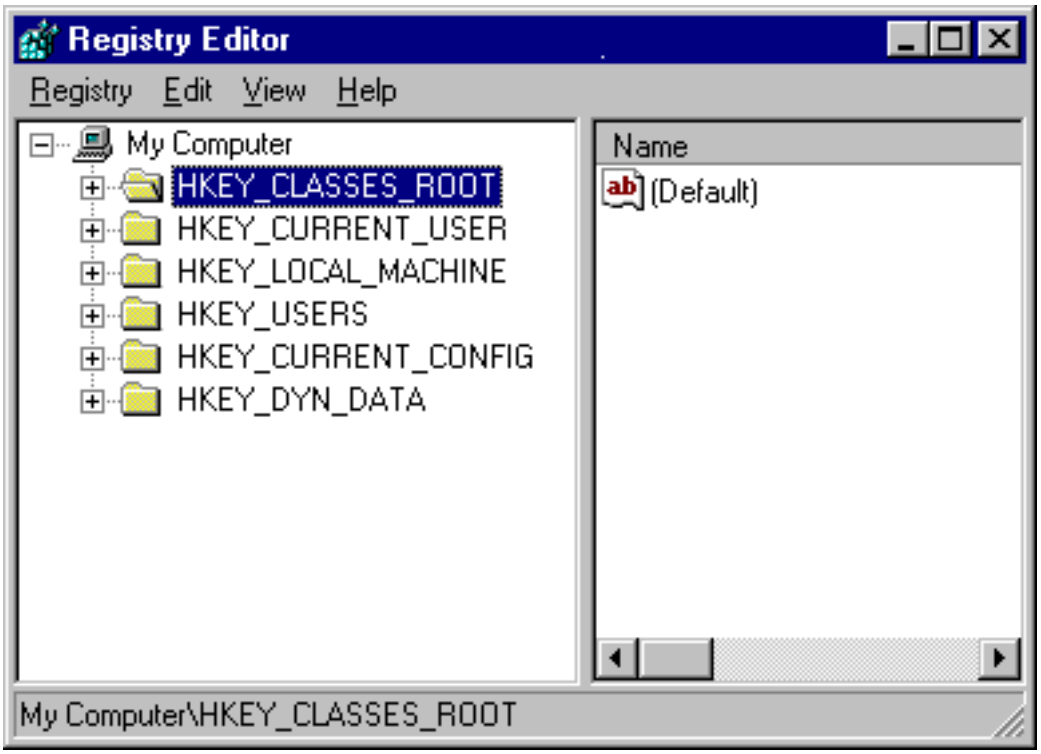


Fig. 1. Registry editor.

The sturcture of COM related information in the registry

Now it's time to launch the registry editor (by entering "regedit" in the command prompt; see fig. 1) and to begin to examine these questions in practice. COM related data are joined in 5 groups: ProgID, CLSID, TypeLib, Interface, AppID. The majority of them have their own registry keys. The root key "HKEY_CLASSES_ROOT" stands for ProgID section. Here we can find the names "Excel.Application" and "Word.Application" that we had used earlier in our scripts together with the remaining 4 COM sections and with file extensions.

ProgID is so called program identifier, it is in fact only "user friendly" label for the "real" name of a component - CLSID. It is accepted that ProgID cannot ensure the true global uniqueness of the component name especially in the distributed internetwork environment so in the latter case only CLSIDs must be used as component identifiers. But on a local machine ProgID is widely used especially in scripts. However the component's CLSID is used for identification of the component in any case; for that crossreferences are established between CLSID and ProgID keys for the component.

Let's examine the key "Excel.Application" for example. It has a subkey CLSID, the default value of which is the required CLSID in a string representation. It means that "HKEY_CLASSES_ROOT\CLSID" contains the corresponding key storing all necessary information for loading the component. Certainly you have paid attention to another key near "Excel.Application" but with a number at the end (the number may differ on various machines; it is "Excel.Application.5" on my machine, for example). The key itself contains another subkey - CurVer. In fact it's "Excel.Application.5" that is ProgID and "Excel.Application" is so called VersionIndependentProgId - "component identifier, independent from the component version" (exactly these subkeys for crossreferences you can find in "HKEY_CLASSES_ROOT\CLSID" key for the corresponding component). Thus it is possible not to indicate the specific version of an installed

component; and if in the course of time the component was upgraded, old scripts would successively work with the new versions if they had used VersionIndependentProgIDs as object names. The COM system has auxiliary funcions CLSIDFromProgID and ProgIDFromCLSID for getting one key using another.

In order to better understand this theory we can have some fun (but not forgetting warnings above). Let's make sure that the "real" name of a component is actually CLSID. For that create a new key under "HKEY_CLASSES_ROOT" and designate it with your own name for variety. Then copy the CLSID value under ProgID "Excel.Application" and paste it as the default value for our newly created CLSID. And now rewrite our script a bit:

```
Set x = CreateObject("MyName")
x.Visible = True
```

Naturally instead of "MyName" you must paste the name chosen for your ProgID. If all Ok Excel now readily responds to the new name.

The utility for ProgID

Examining the registry you can find a great number of interesting thins. Howerer, manual retrieving data following crossreferences is very boring especially when too many components. Combining pleasantness with the profit let's create for our searches a small utility that will translate given ProgID of the component to its CLSID and vice versa.

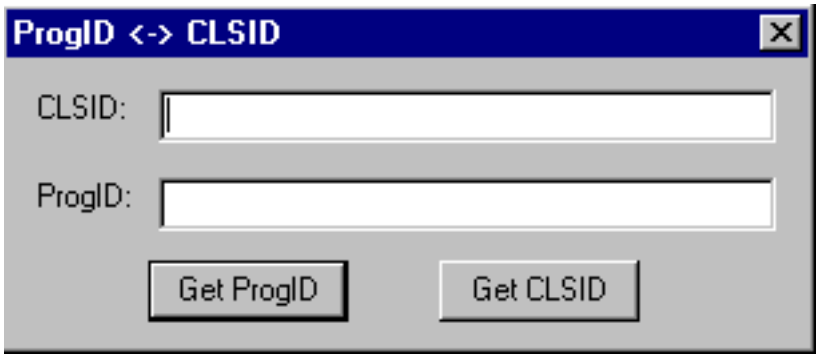


Fig.2. The user interface of COM_1 utility

The user interface design of the utility is shown in fig.2. It's implemented in the following resource file (COM_1.rc):

```
#define DS_MODALFRAME       0x80L
#define DS_CENTER           0x0800L
#define WS_POPUP            0x80000000L
#define WS_CAPTION          0x00C00000L
#define WS_SYSMENU          0x00080000L
#define ES_AUTOHSCROLL      0x0080L

#define IDC_EDIT1           1000
#define IDC_EDIT2           1001
#define IDB_ProgID          1002
#define IDB_CLSID           1003
#define IDC_STATIC          -1

MyDialog DIALOG DISCARDABLE  200, 200, 200, 66
STYLE DS_MODALFRAME | DS_CENTER | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "ProgID <-> CLSID"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "Get ProgID",IDB_ProgID,35,46,50,14
    PUSHBUTTON      "Get CLSID",IDB_CLSID,108,46,50,14
    LTEXT           "CLSID:",IDC_STATIC,7,7,26,12
    LTEXT           "ProgID:",IDC_STATIC,7,27,26,12
    EDITTEXT        IDC_EDIT1,38,7,154,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT2,38,27,154,12,ES_AUTOHSCROLL
END
```

The program's source code is in the file COM_1.asm. The beginning is standard; COM API funcions are located in OLE32.dll so you must include files supporting this module.

```
.386
.model flat,stdcall
option casemap:none

DlgProc proto :DWORD,:DWORD,:DWORD,:DWORD

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\ole32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\ole32.lib

.data

DlgName db "MyDialog",0
App  db "ProgID_CLSID",0
ms1  db "Enter CLSID here",0
ms2  db "Enter ProgID here",0
ms3  db "There is no ProgID for this CLSID",0
ms4  db "There is no CLSID for this ProgID",0
Err1  db "The wrong CLSID",0

.data?

hInstance HINSTANCE ?
hEdit1 DWORD ?  ; edit control for CLSID
hEdit2 DWORD ?  ; edit control for ProgID
buf  db 256 dup(?) ; for ANSI strings
wbuf  db 512 dup(?) ; for Unicode strings
cls  db 16 dup(?) ; buffer for CLSID
bufaddr DWORD ?

.const

IDC_EDIT1 equ 1000 ; edit control for CLSID
IDC_EDIT2 equ 1001 ; edit control for ProgID
IDB_ProgID equ 1002 ; "Get ProgID" button
IDB_CLSID equ 1003 ; "Get CLSID" button
```

The utility is implemented as a modal dialog box created from resource template:

```
.code

start:
invoke GetModuleHandle, NULL
mov    hInstance,eax
invoke DialogBoxParam, hInstance, ADDR DlgName,NULL,
addr  DlgProc, NULL
invoke ExitProcess,eax

DlgProc proc USES esi edi ebx,hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
.IF uMsg==WM_INITDIALOG
 ; store handles for edit controls
 invoke GetDlgItem,hWnd,IDC_EDIT1
 mov hEdit1,eax
 invoke GetDlgItem,hWnd,IDC_EDIT2
 mov hEdit2,eax
 invoke SetFocus,hEdit1
 mov eax,FALSE
 ret
.ELSEIF uMsg==WM_CLOSE
 invoke EndDialog,hWnd,0
```

The algorithm is like that. On pressing "Get ProgID" button edit control for ProgID is cleared and the CLSID control text is read.

```
.ELSEIF uMsg==WM_COMMAND
 mov eax,wParam
 mov edx,wParam
 shr edx,16
 .if dx==BN_CLICKED
  .if ax==IDB_ProgID ; convert CLSID into ProgID
   mov buf,0
   invoke SetWindowText,hEdit2,ADDR buf
   invoke GetWindowText,hEdit1,ADDR buf,255
```

If the string is empty a message suggesting to enter the text is displayed:

```
.if eax==0 ; GetWindowText - the string is empty
 invoke MessageBox,0,ADDR ms1,ADDR App,0
 invoke SendMessage,hEdit1,EM_SETSEL,0,-1
 invoke SetFocus,hEdit1
.else  ; we've got the text
```

A bit of surprise here - COM uses Unicode strings. So we'll have to convert the text in the buffer buf using MultiByteToWideChar function. This function takes 6 parameters:

• code page (in our case - ANSI, the parameter is CP_ACP);
• flags indicating how to translate precomposed or composite wide characters; we don't need it - 0;
• address of string to map;
• number of bytes in string; if -1 the string is null terminated and the length is calculated automatically;
• address of wide-character buffer;
• size of wide-character buffer.

```
 invoke MultiByteToWideChar,CP_ACP,0,ADDR buf,-1,ADDR wbuf,510
```

One more complication. The function ProgIDFromCLSID accepts as an argument the address of a structure with the numeric representation of CLSID and not with the string representation. In more detail we'll discuss this nuance in the section about CLSID and now to convert the string representation into numeric one we'll use one more auxiliary COM API function - CLSIDFromString (where cls is a 16 bytes buffer left for CLSID structure):

```
invoke CLSIDFromString,ADDR wbuf,ADDR cls
.if eax==NOERROR
 invoke ProgIDFromCLSID,ADDR cls,ADDR bufaddr
 .if eax==S_OK
```

In bufaddr we've got string representation of ProgID in Unicode format, now we need to convert it back to ANSI using the function WideCharToMultiByte ("twin" of MultiByteToWideChar). It takes 8 arguments:

• code page we need to convert the Unicode string to; in our case - again CP_ACP;
• performance and mapping flags. We don't need them, so leave 0;
• address of Unicode string;
• number of characters in Unicode string. If -1 the string is null-terminated and the length is calculated automatically;
• address of buffer for the translated string;
• size of buffer for the translated string;
• address of default symbol - we don't need it, leave 0;
• address of flags for default symbol - 0.

```
  invoke WideCharToMultiByte,CP_ACP,0,bufaddr,-1,ADDR buf,255,0,0
  invoke SetWindowText,hEdit2,ADDR buf
 .else ; couldn't convert  CLSID to ProgID -
  ; issue message
  invoke SetWindowText,hEdit2,ADDR ms3
 .endif
```

Further the block for handling errors of function CLSIDFromString goes which are considered as a wrong format of an entered CLSID string:

```
  .else ; error of CLSIDFromString
   invoke MessageBox,0,ADDR Err1,ADDR App,MB_OK OR MB_ICONERROR
   invoke SendMessage,hEdit1,EM_SETSEL,0,-1
   invoke SetFocus,hEdit1
  .endif ;NOERROR - CLSIDFromString
 .endif ;GetWindowText
```

The block handling pressing "Get ProgID" button is complete. Pressing "Get CLSID" button is handled in a similar way.

```
 .elseif ax==IDB_CLSID ;convert ProgID in CLSID string
  mov buf,0
  invoke SetWindowText,hEdit1,ADDR buf
  invoke GetWindowText,hEdit2,ADDR buf,255
  .if eax==0   ; no text in control - error message
   invoke MessageBox,0,ADDR ms2,ADDR App,0
   invoke SendMessage,hEdit2,EM_SETSEL,0,-1
   invoke SetFocus,hEdit2
  .else    ; got ProgID text - convert to Unicode
   invoke MultiByteToWideChar,CP_ACP,0,ADDR buf,-1,ADDR wbuf,510
      ; convert ProgID into numeric CLSID
   invoke CLSIDFromProgID,ADDR wbuf,ADDR cls
   .if eax==S_OK ; got CLSID,
      ; convert into stirng form
    invoke StringFromCLSID,ADDR cls,ADDR bufaddr
      ; convert Unicode string into ANSI
    invoke WideCharToMultiByte,CP_ACP,0,bufaddr,-1,
ADDR  buf,255,0,0
    invoke CoTaskMemFree,bufaddr ; free memory,
      ; allocated in call to StringFromCLSID
    invoke SetWindowText,hEdit1,ADDR buf
   .else   ; failed to get CLSID from ProgID
    invoke SetWindowText,hEdit1,ADDR ms4
   .endif  ;S_OK (CLSIDFromProgID)
  .endif   ;GetWindowText
 .endif    ;ax==IDB_ProgID
```

After that the processing of dialog box messages is over:

```
    .endif ;BN_CLICKED
   .ELSE
    mov eax,FALSE
    ret
   .ENDIF ;uMsg
         mov eax,TRUE
         ret
DlgProc endp

end start
```

The files with sources of this utility (COM_1.rc, COM_1.asm, and makefile) are located in COM_1 folder of ComKit1.rar. The makefile is written with the assumption that MASM32 is installed in \masm32 folder on a current disk; if not it's necessary to edit the makefile.

CLSID key

All primary information about components is stored under the registry key "HKEY_CLASSES_ROOT\CLSID". A separate subkey is created for each component, the subkey name being a string representation of component's CLSID. As was previously mentioned CLSID (like other GUIDs, for example, interface identifiers (IID) or type libraries) has two representations: a numeric one and a string one. The numeric form is presented as a 128-bit value. Theoretically this value could be stored, say, in a XMM register; however COM was created in time of 16-bit machines when nobody knew about any XMM-registers. So the following structure was used for representation of this 128-bit value:

```
GUID STRUCT
```

```
    Data1  dd ?
    Data2  dw ?
    Data3  dw ?
    Data4  db 8 dup(?)
GUID ENDS
```

But in the registry a string of the following format is used for representation of GUID:

```
{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}
```

where x stands for a 16-bit digit (0-F), and braces and hyphens must be present in place without any delimiting spaces. Actually such a record can cause confusion: note that the block of digits for Data4 field in the string representation is divided into two parts of 4 and 12 hexadecimal digits, and the part with 4 digits may be confused with the previous WORD-values. The first DWORD and the two following WORD values are recorded in the string representation as usual numbers, i.e. high bytes go first, whereas the third "WORD" (the first two bytes of Data4 field) is recorded in the "little-endian" order - the low byte goes first.

As we've already seen in the first utility source the functions CLSIDFromString and StringFromCLSID are used for converting CLSID from the string representation into the numeric one and vice versa correspondingly. There are choices for other GUID types - StringFromIID, IIDFromString, StringFromGUID2.

CLSID key stores in turn a number of subkeys. We know already two of them - ProgID and VersionIndependentProgID. It's worth mentioning only that the presence of ProgID for a component is not mandatory.

The simple universal COM client

Before we advance in examining CLSID section we'll need one more utility - a universal COM client. Its user interface is showed on fig.3. This client allows to load COM object with CLSID entered in the first edit control and to acquire the interface pointer with IID entered in the second edit control. The server type is indicated with a set of checkboxes (it's possible to check more than one type at once). IUnknown button allows to display the IID of the often used interface with the same name in the second edit control.

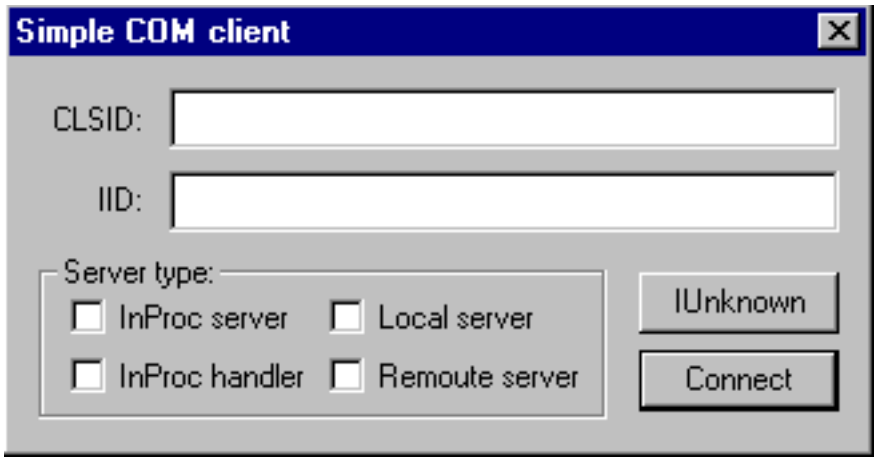


Fig.3. The user interface of COM_2

Here's the corresponding resource file (COM_2.rc):

```
#include "\masm32\include\resource.h"

#define IDC_EDIT1  1001
#define IDC_EDIT2  1002
#define IDC_CHECK1 1003
#define IDC_CHECK2 1004
#define IDC_CHECK3 1005
#define IDC_CHECK4 1006
#define IDC_BUTTON1 1007
```

```
#define IDC_STATIC -1

MyDialog DIALOGEX 0, 0, 214, 90
STYLE DS_MODALFRAME | DS_CENTER | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Simple COM client"
FONT 8, "MS Sans Serif"
BEGIN
    LTEXT           "CLSID:",IDC_STATIC,7,10,26,8,0,WS_EX_RIGHT
    LTEXT           "IID:",IDC_STATIC,7,28,26,8,0,WS_EX_RIGHT
    GROUPBOX        "Server type:",IDC_STATIC,7,45,142,38
    EDITTEXT        IDC_EDIT1,40,7,167,14,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT2,40,26,167,14,ES_AUTOHSCROLL
    CONTROL         "InProc server",IDC_CHECK1,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,15,55,58,10
    CONTROL         "InProc handler",IDC_CHECK2,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,15,68,62,10
    CONTROL         "Local server",IDC_CHECK3,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,80,55,55,10
    CONTROL         "Remoute server",IDC_CHECK4,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,80,68,66,10
    PUSHBUTTON      "IUnknown",IDC_BUTTON1,157,49,50,14
    DEFPUSHBUTTON   "Connect",IDOK,157,67,50,14
END
```

So as not to write out standard defines every time it's advisable to copy "resource.h" from the 10th lesson of Iczelion into "Include" folder of MASM32 and then put into rc-file the proper reference as it's done in this case.

The implementation is in COM_2.asm:

```
.386
.model flat,stdcall
option casemap:none

DlgProc proto :DWORD,:DWORD,:DWORD,:DWORD

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\ole32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\ole32.lib

.data

DlgName db "MyDialog",0
app  db "SimpleClient",0
ms1  db "Enter CLSID here",0
ms2  db "Enter IID here",0
szUnk  db "{00000000-0000-0000-C000-000000000046}",0
mscheck db "The server type must be indicated",0
msclsid db "Enter CLSID here",0
msiid  db "Enter IID here",0
msok db "Got interface pointer to:",13,10,"CLSID:",9,"%s",13,10,"IID:",9,"%s",0
mserr  db "CoCreateInstance failed:",13,10,"HRESULT==%Xh",0
err1  db "The wrong CLSID",0
err2  db "The wrong IID",0

.data?

hInstance HINSTANCE ?
hEdit1 DWORD ?  ; "CLSID"
hEdit2 DWORD ?  ; "IID"
hCheck1 DWORD ?  ; "Inproc server"
hCheck2 DWORD ?  ; "Inproc handler"
hCheck3 DWORD ?  ; "Local server"
hCheck4 DWORD ?  ; "Remoute server"
ctx  DWORD ?  ; server type
szclsid db 64 dup(?) ; buffer for CLSID string
sziid  db 64 dup(?) ; buffer for IID string
```

```
buf  db 256 dup(?) ; for ANSI strings
wbuf  db 512 dup(?) ; for Unicode strings
cls  db 16 dup(?) ; CLSID of component
iid  db 16 dup(?) ; IID of called interface
pUnk  DWORD ?  ; ptr to IUnknown interface

.const

IDC_EDIT1 equ 1001 ; "CLSID"
IDC_EDIT2 equ 1002 ; "IID"
IDC_CHECK1 equ 1003 ; "Inproc server"
IDC_CHECK2 equ 1004 ; "Inproc handler"
IDC_CHECK3 equ 1005 ; "Local server"
IDC_CHECK4 equ 1006 ; "Remoute server"
IDC_BUTTON1 equ 1007 ; "IUnknown"

.code

start:
 invoke GetModuleHandle, NULL
 mov    hInstance,eax
 invoke DialogBoxParam, hInstance, ADDR DlgName,NULL, addr DlgProc, NULL
 invoke ExitProcess,eax

DlgProc proc USES esi edi ebx,hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM
.IF uMsg==WM_INITDIALOG
 ; store handles of controls
 invoke GetDlgItem,hWnd,IDC_EDIT1
 mov hEdit1,eax
 invoke GetDlgItem,hWnd,IDC_EDIT2
 mov hEdit2,eax
 invoke GetDlgItem,hWnd,IDC_CHECK1
 mov hCheck1,eax
 invoke GetDlgItem,hWnd,IDC_CHECK2
 mov hCheck2,eax
 invoke GetDlgItem,hWnd,IDC_CHECK3
 mov hCheck3,eax
 invoke GetDlgItem,hWnd,IDC_CHECK4
 mov hCheck4,eax
 invoke SetFocus,hEdit1
```

Since this time we need to enable the COM library, it's also necessary to call the function CoInitialize. In the end of the program it's necessary to call CoUninitialize correspondingly.

```
 invoke CoInitialize,0
 mov eax,FALSE
 ret
.ELSEIF uMsg==WM_CLOSE
 invoke CoUninitialize
 invoke EndDialog,hWnd,0
```

The core of this utility is made up of CoCreateInstance function that makes all work - locates the proper COM object, loads it, requests the indicated interface and returns a pointer to it. The bulk of code handling WM_COMMAND collects data for CoCreateInstance for passing as its arguments and checks the entered values. CoCreateInstance takes 5 arguments:

• address of structure containing CLSID of the requested object (we get this value in the first edit control);
• pointer to the so called outer IUnknown in the case if an object is being created as part of an aggregate. If not (as in our case) the value is 0;
• context for running code: flags indicating whether an object server is inprocess, local, remoute or it is an inprocess handler (this value is calculated basing on checking the state of checkboxes);
• address of structure containing IID of the interface which pointer we want to get to. The IID value we get in the second edit contol;
• address of pointer variable that receives the requested interface pointer (or 0, if the interface is not implemented).

When pressing IUnknown button IID of the corresponding interface is copied into the second edit control (this value is hardcoded as szUnk string). So, here's handling of commands:

```
.ELSEIF uMsg==WM_COMMAND
 mov eax,wParam
 mov edx,wParam
 shr edx,16
 .if dx==BN_CLICKED
  .if ax==IDC_BUTTON1  ; button 'IUnknown'
invoke SetWindowText,hEdit2,offset szUnk
  .elseif ax==IDOK  ; button 'Connect'
      ; get CLSID
invoke GetWindowText,hEdit1,offset szclsid,64
.if eax==0     ; no text - display message
 invoke MessageBox,0,offset msclsid,offset app,MB_ICONEXCLAMATION
 invoke SendMessage,hEdit1,EM_SETSEL,0,-1
 invoke SetFocus,hEdit1
 mov eax,TRUE
 ret
.else      ; got text - translate into ANSI and
      ; convert to numeric form, storing in cls
invoke MultiByteToWideChar,CP_ACP,0,offset szclsid,-1,offset wbuf,510
 invoke CLSIDFromString,offset wbuf,offset cls
 .if eax!=NOERROR   ; error of CLSIDFromString is considered
      ; as wrong format of entered CLSID string
  invoke MessageBox,0,offset err1,offset app,MB_ICONERROR
  invoke SendMessage,hEdit1,EM_SETSEL,0,-1
  invoke SetFocus,hEdit1
  mov eax,TRUE
  ret
 .endif    ; NOERROR - CLSIDFromString
.endif     ; GetWindowText (hEdit1)

      ; get IID
invoke GetWindowText,hEdit2,offset sziid,64
.if eax==0 ; no text - issue message
 invoke MessageBox,0,offset msiid,offset app,MB_ICONEXCLAMATION
 invoke SendMessage,hEdit2,EM_SETSEL,0,-1
 invoke SetFocus,hEdit2
 mov eax,TRUE
 ret
.else      ; got IID text, translate into ANSI,
      ; then convert to numeric form, storing in iid
 invoke MultiByteToWideChar,CP_ACP,0,offset sziid,-1,offset wbuf,510
 invoke IIDFromString,offset wbuf,offset iid
 .if eax!=NOERROR   ; error of IIDFromString is considered as
      ; wrong format of entered IID string
  invoke MessageBox,0,offset err2,offset app,MB_ICONERROR
  invoke SendMessage,hEdit2,EM_SETSEL,0,-1
  invoke SetFocus,hEdit2
  mov eax,TRUE
  ret
 .endif    ; NOERROR - IIDFromString
.endif     ; GetWindowText (hEdit2)

  ; chech the state of checkboxes and determine the type of server,
  ; storing the calculated value in ctx
mov ctx,0
invoke IsDlgButtonChecked,hWnd,IDC_CHECK1
.if eax==BST_CHECKED
 or ctx,CLSCTX_INPROC_SERVER
.endif
invoke IsDlgButtonChecked,hWnd,IDC_CHECK2
.if eax==BST_CHECKED
 or ctx,CLSCTX_INPROC_HANDLER
.endif
invoke IsDlgButtonChecked,hWnd,IDC_CHECK3
.if eax==BST_CHECKED
 or ctx,CLSCTX_LOCAL_SERVER
.endif
invoke IsDlgButtonChecked,hWnd,IDC_CHECK4
```

```
.if eax==BST_CHECKED
 or ctx,CLSCTX_REMOTE_SERVER
.endif
.if ctx==0    ; no server type is selected - display message
 invoke MessageBox,0,offset mscheck,offset app,MB_ICONEXCLAMATION
 invoke SetFocus,hCheck1
 mov eax,TRUE
 ret
.endif
```

Now the necessary parameters are collected: CLSID is in cls, IID - in iid, the server type - in ctx. CoCreateInstance locates COM object with the given CLSID and instantiates it, requesting pointer to IID on it. If the object implements an interface with the given IID, it places the pointer to it in a variable, address of which was passed to CoCreateInstance as the last parameter, and the function itself returns S_OK, If the object doesn't implement this interface or an object with the given CLSID isn't registered in the registry, or if the registered server type isn't indicated in the flag of server type, an error message is returned.

```
invoke CoCreateInstance,offset cls,0,ctx,offset iid,offset pUnk
mov ecx,eax
.if eax==S_OK ; The object is successively created and a pointer to the
  ; requested interface is returned in  pUnk. Issue message
  ; with CLSID of the object and IID of the interface
 invoke wsprintf,offset buf,offset msok,offset szclsid,offset sziid
 invoke MessageBox,0,offset buf,offset app,MB_ICONINFORMATION
```

At this place all "useful" work of our application is up - we free the object. It's necessary to do it using COM facilities, i.e. using obtained interface pointer. For that it's necessary to call a Release method of the IUnknown interface using the obtained interface pointer (pUnk). Absolutely all COM interfaces are inherited from IUnknown, therefore the Release method may be called by means of the same technique for all COM objects.

A pointer to an object interface is an address of a structure storing data related to the state of an object instance. Only indirect access to this data is possible by calling methods of the proper interface so each method receives an address of this structure (a so called "this" pointer in terms of C++ language) as the first parameter. It's possible to say about this structure only that its first field stores a pointer to another structure - "virtual table" - containing addresses of functions implementing methods of the interface. To call this or that method one needs to find an address of its implementation. The address of the method is stored in the virtual table with the well-known and constant offset. Machinery of this process we'll examine in more detail in the next article and now just the code:

```
 mov eax,pUnk
 push eax  ; ptr to object instance ("this")
 mov eax,[eax]  ; get ptr to virtual table,
 call dword ptr [eax+8] ; then ptr to the third function
    ; in this table (Release) and call it

.else   ; error of CoCreateInstance
 invoke wsprintf,offset buf,offset mserr,ecx
 invoke MessageBox,0,offset buf,offset app,MB_ICONERROR
.endif   ; eax==S_OK (CoCreateInstance)
```

At this place handling of WM_COMMAND and of any other messages is completed.

```
    .endif ;ax==IDOK

   .endif ;BN_CLICKED

  .ELSE ; uMsg is not handled
   mov eax,FALSE
   ret
  .ENDIF ; uMsg
        mov eax,TRUE
        ret
DlgProc endp
```

```
end start
```

Subkeys of CLSID key

Now it's possible to continue experiments with the registry. The major subkeys of "HKEY_CLASSES_ROOT\CLSID" are a number of subkeys allowing to locate a server in the system: InprocServer, InprocServer32, InprocHandler, InprocHandler32, LocalServer, LocalServer32. Each of the server types is represented with two subkeys; the suffix "32" means that the server is 32-bit, the subkey without this suffix indicates that the server is 16-bit (for backward compatibility). The case of the remote server is handled in a particular way; we'll discuss it in more details in the section about AppID.

Let's begin to play tricks. Special utilities such as guidgen etc. are used for getting unique GUIDs when creating new interfaces and components. Microsoft has reserved a large number of handy GUIDs with sets of zeros for its own use; but we'll handle it yet cooler - we'll use GUIDs like {00000000-0000-0000-0000-000000000001}. Why to bother ourselves seeking for our component among the bulk of digits, when we can comfortably put it in the very beginning of the section? It would be possible (and the author had done it indeed at first) to use GUID with all zeros but it's the special reserved GUID_NULL used in special cases; to avoid trouble it's better to leave it alone.

Create the key {00000000-0000-0000-0000-000000000001} under "HKEY_CLASSES_ROOT\CLSID". The key may have a default value which represents the textual description of the component for a user. Note that it isn't ProgID (though some component programmers use ProgID as the component description). You may put here some string, for example, "My favourite component". Then create a subkey with the name "LocalServer32". The default value of this subkey stores just the full pathname to the file implementing the server. Write here "c:\windows\calc.exe" (replace "windows" with the windows folder on your machine). Now launch our client (COM_2.exe), enter CLSID, press IUnknown, check "Local server" as the server, press "Connect" and... You may close the calculator. And our client is like hung - but you must have been ready to this, because the calculator has no idea about such an occasion that it became a COM component. While the COM system is still waiting for a responce from the calculator, our client sits inside CoCreateInstance at that time. Though about half a minute later it'll return the timeout error and our client will display the corresponding message box.

I shan't describe here all other experiments - the way is showed, the utilities are created - experiment! COM technology may be understood only in practice. But don't forget about the warning concerning accuracy when working with registry in the heat of enthusiasm.

Let's consider other subkeys. The important ones are TreatAs and AutoTreatAs because they allow to change the server of a component. If there is a subkey TreatAs (its default value contains another CLSID) under the key with the given CLSID then an object will be created with this other CLSID regardless of presence of subkeys InprocServer, LocalServer, etc. Make the following experiment. Create a subkey "TreatAs" under our key {00000000-0000-0000-0000-000000000001}. Put CLSID for the component Excel.Application (using the utility COM_1.exe to convert ProgID to CLSID) as the value of this subkey. Now if you'll try to load our component using COM_2.exe, Excel will be loaded (though it remains hidden. To make sure it runs look into the task manager).

The subkey TreatAs permits one server to emulate the other one. The function CoGetTreatAsClass allows to get a value of this key and the function CoTreatAsClass - to assign this value or delete it. If the subkey TreatAs is being deleted using CoTreatAsClass this function examines the value of the subkey AutoTreatAs. If a key with this name does exist its value is copied in the subkey TreatAs; if not the TreatAs subkey is deleted alltogether. And the subkey AutoTreatAs may be created or deleted only by hand using registry API functions. Thus the value of preceding CLSID may be stored in the AutoTreatAs subkey if this server was emulated at first by one server and then is emulated by the other providing in a sense "permanent" emulation (as opposed to "temporary" emulation using TreatAs subkey).

Being the main source of data about the component the "HKEY_CLASSES_ROOT\CLSID" key contains subkeys with crossreferences to other keys. We've already talked about ProgID and

VersionIndependentProgID subkeys. The TypeLib subkey stores GUID that is an identifier of the type library for the given component; the corresponding subkey is located under "HKEY_CLASSES_ROOT\TypeLib". AppID subkey stores GUID identifying the subkey of another section - "HKEY_CLASSES_ROOT\AppID" - and indicates that the object is distributed (DCOM). These subkeys will be described later in more details.

There is a few other subkeys; we shan't examine them all. We'll mention only about those that are in a sense flags indicating types of components. The presence of "Control" subkey indicates that the component is a control; "Insertable" indicates that the object can be embedded in OLE containers; "OLEScript" - that the object is a scripting engine; "DocObject" - a document object; "Printable" - the object can be printed; "Programmable" - the object is an automation server (and accordingly it can be managed with scripts); "Ole1Class" - OLE 1.0 object (old version).

The utility for examination of object types

Let's create another small utility that will display a list of objects registered in the registry and relating to one of seven types listed at the end of the previous paragraph. The user interface of the utility is shown in fig.4. The type of an object can be chosen in the dropdown list; after pressing the "List" button descriptions of proper objects are displayed in the listview (these descriptions are default values for corresponding CLSID keys). If there is no description the string will be left empty.

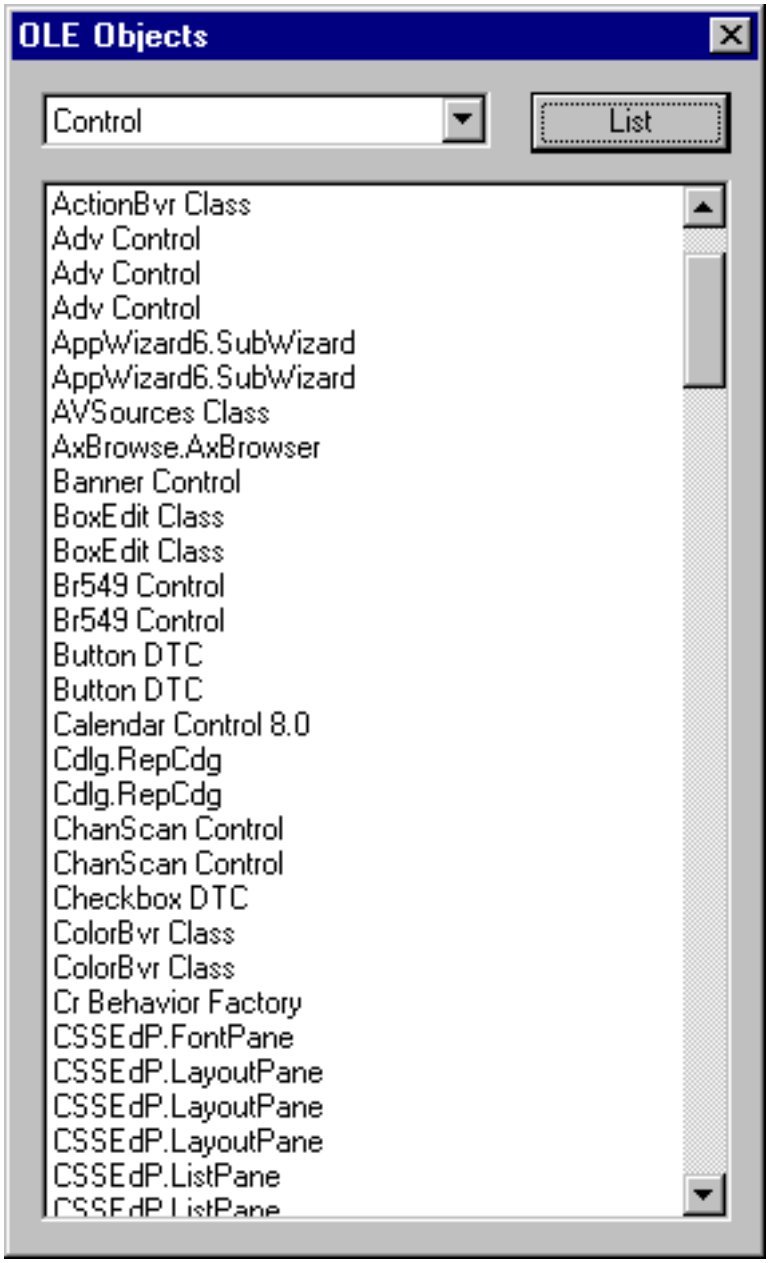


Fig.4. The user interface of COM_3 utility

To save space we shan't give here the full source code of the application; the respective files are located in COM_3 folder of ComKit1.rar. Instead let's examine only some of essential aspects concerning the logic of program operation.

```
invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey1,0,KEY_ALL_ACCESS,ADDR HKey
.if eax==ERROR_SUCCESS
```

First the registry key "HKEY_CLASSES_ROOT\CLSID" is opened; its handle is stored in HKey variable. On successful opening of the key the main loop starts enumerating subkeys stored in the opened key.

```
EnumLoop:
 mov sbksz,255
 invoke RegEnumKeyEx,HKey,idx,ADDR SubKeyBuf,ADDR sbksz,0,0,0,0
 .if eax==ERROR_NO_MORE_ITEMS
  jmp OutLoop
 .elseif eax!=ERROR_SUCCESS
  invoke MessageBox,0,ADDR Err2,ADDR App,MB_OK OR MB_ICONERROR
  jmp OutLoop
 .endif
```

On any enumeration error the loop is being broken and if the error is not because the enumeration is up, the proper message is displayed. For each new key found the two strings are constructed: "CLSID\{clsid of component}" and "CLSID\{clsid of component}\<type of object>", where <type of object> represents one of the seven strings describing the type of the object chosen from the dropdown list.

```
  invoke lstrcpy,ADDR SubKey2,ADDR SubKey1
  invoke lstrcpy,ADDR SubKey3,ADDR SubKey1
  invoke lstrcat,ADDR SubKey2,ADDR s
  invoke lstrcat,ADDR SubKey3,ADDR s
  invoke lstrcat,ADDR SubKey2,ADDR SubKeyBuf
  invoke lstrcat,ADDR SubKey3,ADDR SubKeyBuf
  invoke lstrcat,ADDR SubKey3,ADDR s
  invoke lstrcat,ADDR SubKey3,ADDR findbuf
  invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey3,0,
KEY_ALL_ACCESS,ADDR HKey2
.if eax==ERROR_SUCCESS
```

Then an attempt is made to open the registry key "CLSID\{clsid of component}\<type of object>". The successful attempt means that the key with the given CLSID has a subkey with the chosen name; in this case the other prepared key ("CLSID\{clsid of component}") is opened and the default value is requested which is then sent to the listview and the loop continues with the next iteration:

```
  invoke RegCloseKey,HKey2
  invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey2,0,
KEY_ALL_ACCESS,ADDR HKey3
  invoke RegQueryValueEx,HKey3,0,0,0,ADDR namebuf,ADDR bsz
  invoke RegCloseKey,HKey3
  invoke SendMessage,hLst,LB_ADDSTRING,0,ADDR namebuf
.endif ;RegOpenKeyEx (2)
  inc idx
  jmp EnumLoop
OutLoop:
  invoke RegCloseKey,HKey
```

Interface section

Unlike other keys "HKEY_CLASSES_ROOT\Interface" relates not to individual components but to the system in the whole. Given here data about interfaces is used on standard marshaling. There are not so many subkeys: BaseInterface, NumMethods, ProxyStubCLSID, and ProxyStubCLSID32. BaseInterface stores the name of the interface which the given interface is inherited from; if it isn't indicated IUnknown is used as the base interface. NumMethods stores the number of methods in the interface. ProxyStubCLSID(32) is like InprocServer(32): it stores the full pathname to the server (dll) implementing auxiliary objects (so called proxies and stubs), which are used by the systm on marshaling of methods of the given interface. As usual the suffix "32" in the name means the 32-bit server and its absence means the 16-bit one.

Detailed discussion of marshaling and related problems is beyond the scope of this article; something will be told in the second article, and now we'll give the description of a useful utility using this registry section. Its user interface is shown in fig.5. On pressing "List" button the application creates an instance of the COM object with CLSID indicated in the edit box. Then the created object is requested for all possible interfaces names of which can be found in the system registry. The names of all interfaces implemented by this object are displayed in the listview.

Fig.5. The user interface of COM_4 utility

Here again only key features of the cource code will be given; source files are located in the folder COM_4 of ComKit1.rar.

Since the application uses the COM library CoInitialize procedure is called on initilizing of the dialog box and on its closing CoUninitialize is called appropriately. On pressing "List" button the listview window is cleared and CLSID is got from editbox by means we already know:

```
invoke GetWindowText,hEd,ADDR buffer,255
invoke MultiByteToWideChar,CP_ACP,0,ADDR buffer,511,ADDR wbuf,255
invoke CLSIDFromString,ADDR wbuf,ADDR cls
```

Then an instance of the object with the given CLSID is created; IUnknown is requested as the first interface (its IID is stored in IUnk) and the object context permits any type of the server:

```
invoke CoCreateInstance,ADDR cls,0, CLSCTX_SERVER,ADDR IUnk,ADDR pUnk
```

If the pointer is obtained (in pUnk) successfully the registry key "HKEY_CLASSES_ROOT\Interface" is opened and the main loop of enumeration of subkeys (i.e. interfaces) is entered:

```
invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey1,0,
```

```
KEY_ALL_ACCESS,ADDR HKey
 .if eax==ERROR_SUCCESS
Enum1:
 mov sbksz,255
 invoke RegEnumKeyEx,HKey,idx,ADDR SubKeyBuf,ADDR sbksz,0,0,0,0
 .if eax==ERROR_NO_MORE_ITEMS
  jmp Ex_2
 .elseif eax!=ERROR_SUCCESS
  invoke MessageBox,0,ADDR Err2,ADDR App,MB_OK OR MB_ICONERROR
  jmp Ex_2
```

The loop is exited when the enumeration function returns an error; if it isn't the error because of lack of keys for further enumeration then an error message is displayed. Each new enumerated key (IID of an interface in the string representation) is converted into numeric form (if an error takes place when converting then this key subsequently is simply ignored and the loop continues).

```
 .else
  invoke MultiByteToWideChar,CP_ACP,0,ADDR SubKeyBuf,255,ADDR wbuf,255
  invoke CLSIDFromString,ADDR wbuf,ADDR cls
  .if eax!=NOERROR
   jmp Cont1
  .endif
```

Now it's necessary to request our object for the interface with the just got IID. To do it we need to call QueryInterface method of IUnknown interface with the following arguments:

• pointer to an instance of the current object ("this");
• address of a structure where IID of the requested interface is stored;
• address of output variable that receives the requested interface pointer.

Address of the QueryInterface method is located at the very beginning of the virtual table (with offset 0):

```
 lea eax,pItfc ; pItfc receives ptr to new interface
 push eax
 lea eax,cls  ; requested IID is in cls
 push eax
 mov eax,pUnk  ; ptr to IUnknown interface
    ; of our object ("this")
 push eax
 mov eax,[eax]  ; get address of the virtual table
 call dword ptr [eax] ; call the first function
    ; from the virtual table (QueryInterface)
```

In the case of successful receiving of the new interface pointer QueryInterface returns S_OK. We need just the fact that the object implements this interface, the interface itself is not required; therefore we release just received pointer right away calling Release method on it:

```
.if eax==S_OK
 mov eax,pItfc  ; ptr to requested interface
 push eax  ; put as the first (and the only) parameter;
 mov eax,[eax]  ; address of virtual table
 call dword ptr [eax+8] ; call the 3rd method from the table (Release)
```

At the same time a kind of "Interface\{IID of interface}" string is created and the respective registry key is opened requesting the default value of this key (the interface name). This name is added into the listview.

```
 invoke lstrcpy,ADDR SubKey2,ADDR SubKey1
 invoke lstrcat,ADDR SubKey2,ADDR s
 invoke lstrcat,ADDR SubKey2,ADDR SubKeyBuf
 invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey2,0,
KEY_ALL_ACCESS,ADDR HKey2
 .if eax==ERROR_SUCCESS
  mov bsz,255
  invoke RegQueryValueEx,HKey2,0,0,0,ADDR namebuf,ADDR bsz
  .if eax!=ERROR_SUCCESS
   invoke MessageBox,0,ADDR Err5,ADDR App,MB_OK OR MB_ICONERROR
  .endif
  invoke RegCloseKey,HKey2
```

```
    invoke SendMessage,hLst,LB_ADDSTRING,0,ADDR namebuf
   .endif ;RegOpenKeyEx (2)
  .endif ;S_OK (QueryIntervace)
```

In case if QueryInterface or RegOpenKeyEx returns error this key is simply ignored and the loop continues.

```
  Cont1:
   inc idx
   jmp Enum1
   .endif ;RegEnumKeyEx
  Ex_2:
   invoke RegCloseKey,HKey
```

After exiting the loop Release method is called on pUnk pointer to IUnknown interface that was received on creating the object instance (in a similar manner it was done earlier).

TypeLib section

"HKEY_CLASSES_ROOT\TypeLib" contains keys the names of which represent the string form of identifiers (GUIDs) of istalled in the system type libraries. Subkeys TypeLib of components under "HKEY_CLASSES_ROOT\CLSID" reference these keys but there are no backreferences. Type libraries are mandatory for components that use late binding and automation; in particular they are widely used with ActiveX controls. Other components may not have type libraries.

Each key under TypeLib contains a hierarchical structure of subkeys. Subkeys of type library version are stored on the first level, being represented in the string form like major.minor (digits corresponding to the version). In turn a version subkey contains HelpDir subkeys (that store full pathnames to help files), Flags (flag of type library) and the string representation of the locale identifier (lcid). The <lcid> key contains yet one or two additional subkeys: win16 and/or win32 - full pathnames to type libraries for the corresponding platform.

Let's consider one more application as an example of work with a type library (fig.6). The dialog box of the application has only one control - a listview. Descriptions of components are displayed in the first column and identifiers of the corresponding type libraries are displayed next to them in the second column. On double click on component name either general info about the given component stored in its type library is displayed or help file related to this component (if any) is opened.

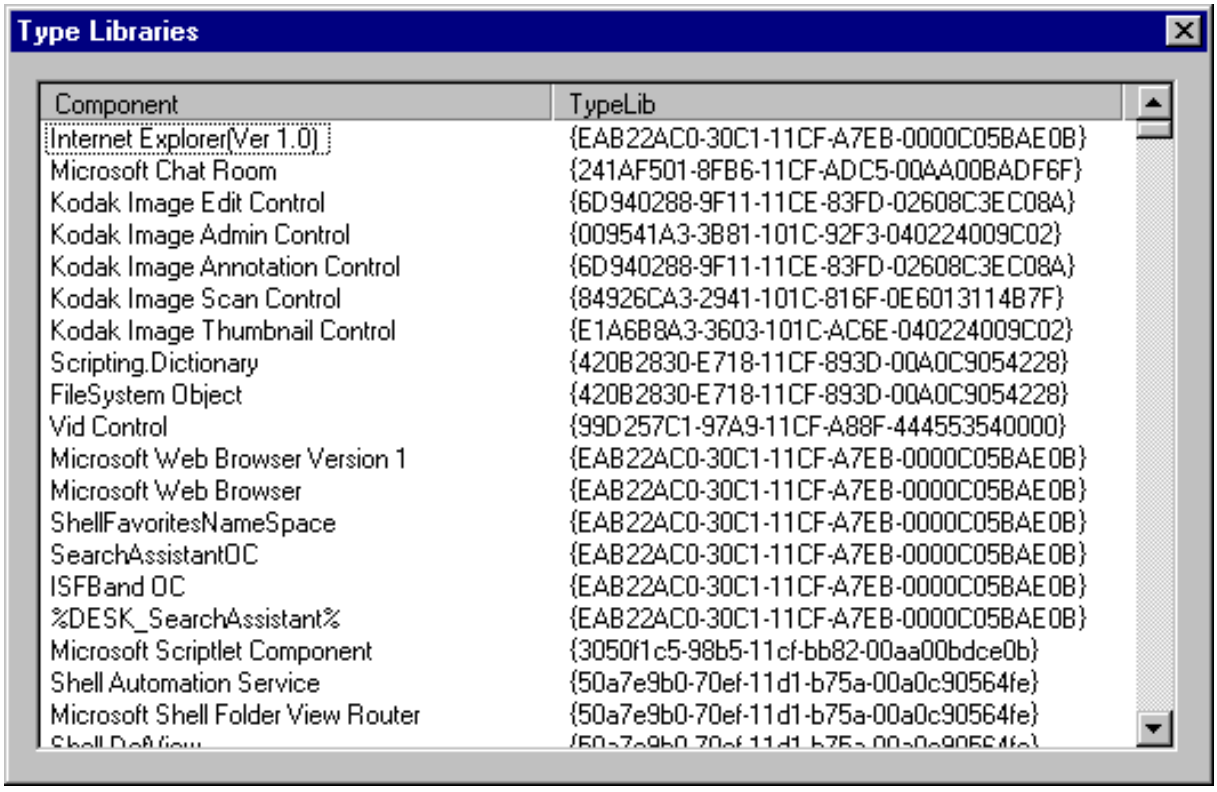


Fig.6. The user interface of COM_5 utility

The application works as follows (details of implementation of GUI are omitted; full source code is located in COM_5 folder of ComKit1.rar). On handling WM_INITDIALOG message the registry key "HKEY_CLASSES_ROOT\CLSID" is opened, then the main loop enumerating stored there subkeys is entered:

```
invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey1,0,KEY_ALL_ACCESS,ADDR HKey
 .if eax==ERROR_SUCCESS
EnumLoop:
  lea esi,buf
  mov sbksz,255
  invoke RegEnumKeyEx,HKey,idx,ADDR SubKeyBuf,ADDR sbksz,0,0,0,0
  .if eax==ERROR_NO_MORE_ITEMS
   jmp OutLoop
  .elseif eax!=ERROR_SUCCESS
   invoke MessageBox,0,ADDR Err2,ADDR App,MB_OK OR MB_ICONERROR
   jmp OutLoop
  .endif
```

As in the previous utiltity if RegEnumKeyEx returns an error the loop is being broken. The two strings in the form of "CLSID\{clsid of component}" and "CLSID\{clsid of component}\TypeLib" are created for each new key, then an attempt is made to open the registry key with the second string as the name.

```
 invoke lstrcpy,ADDR SubKey2,ADDR SubKey1
 invoke lstrcpy,ADDR SubKey3,ADDR SubKey1
 invoke lstrcat,ADDR SubKey2,ADDR s
 invoke lstrcat,ADDR SubKey3,ADDR s
 invoke lstrcat,ADDR SubKey2,ADDR SubKeyBuf
 invoke lstrcat,ADDR SubKey3,ADDR SubKeyBuf
 invoke lstrcat,ADDR SubKey3,ADDR s
 invoke lstrcat,ADDR SubKey3,ADDR findbuf
 invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey3,0,
KEY_ALL_ACCESS,ADDR HKey3
```

The successful opening of TypeLib key means the component has a type library. In this case a value of "CLSID\{clsid of component}" key is requested and the corresponding string is placed in the first column of the listview and a value of "CLSID\{clsid of component}\TypeLib" subkey (the string representation of the

type library identifier) is placed in the second column:

```
.if eax==ERROR_SUCCESS
invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,
ADDR SubKey2,0,KEY_ALL_ACCESS,ADDR HKey2
 .if eax==ERROR_SUCCESS
mov bsz,255
invoke RegQueryValueEx,HKey2,0,0,0,ADDR buf,ADDR bsz
  mov eax,cnt
mov item.iItem,eax
mov item.iSubItem,0
invoke SendMessage,hList,LVM_INSERTITEM,0,ADDR item
 mov bsz,255
 invoke RegQueryValueEx,HKey3,0,0,0,ADDR buf,ADDR bsz
  .if eax==ERROR_SUCCESS
   mov item.iSubItem,1
   invoke SendMessage,hList,LVM_SETITEM,0,ADDR item
  .endif
  invoke RegCloseKey,HKey2
  inc cnt
 .endif ;RegOpenKeyEx (SubKey2)
 invoke RegCloseKey,HKey3
.endif ;RegOpenKeyEx (SubKey3)
inc idx
jmp EnumLoop
OutLoop:
 invoke RegCloseKey,HKey
```

A double click on an element of the listview is handled as WM_NOTIFY message. The proper code is located in the separate procedure FindHelp, the address of NMLISTVIEW structure with info about the clicked list element being passed in ebx registry:

```
.ELSEIF uMsg==WM_NOTIFY
 mov ebx,lParam
 assume ebx:ptr NMHDR
 .if [ebx].code==NM_DBLCLK
  call FindHelp
 .endif
 assume ebx:nothing
```

FindHelp procedure begins with checking whether double click is acually made on a listview element:

```
FindHelp proc
 assume ebx:ptr NMLISTVIEW
 mov eax,[ebx].iItem
 mov item.iItem,eax
 .if eax==-1
  ret
 .endif
 assume ebx:nothing
```

Now it's possible to copy into buf contents of the second column of the proper element (GUID of the type library in the text representation).

```
 mov item.imask,LVIF_TEXT
 mov item.iSubItem,1
 mov item.pszText,OFFSET buf
 mov item.cchTextMax,255
 invoke SendMessage,hList,LVM_GETITEM,0,ADDR item
```

Then using the given value the name of the registry key "TypeLib\{GUID of type library}" is created and the key is opened.

```
 invoke lstrcpy,ADDR SubKey2,ADDR cl2
 invoke lstrcat,ADDR SubKey2,ADDR s
 invoke lstrcat,ADDR SubKey2,ADDR buf
 invoke RegOpenKeyEx,HKEY_CLASSES_ROOT,ADDR SubKey2,
0,KEY_ALL_ACCESS,ADDR HKey
```

It's necessary to handle the not very handy form of representation of type library data. To get a pathname to the file containing the type library, first we need to indicate the version but it's impossible to say anything about it in advance. The trouble is that the function LoadRegTypeLib requires indicating of the exact major version and refuses to load type libraries with other versions. Therefore we'll need to examine version subkeys on hand and pick out the major version from the subkey name.

```
  .if eax==ERROR_SUCCESS
   mov idx,0
   mov mjver,0 ; for max value of majversion
TlibLoop:
   mov sbksz,255
   ; enumerate subkeys under 'HKCR\TypeLib\{GUID}'
   ; (as majversion.minversion)
   invoke RegEnumKeyEx,HKey,idx,ADDR SubKeyBuf,ADDR sbksz,0,0,0,0
   .if eax==ERROR_SUCCESS
    xor eax,eax
    mov al,SubKeyBuf ; the first digit of subkey (i.e. majversion)
    sub eax,30h  ; convert the symbol into value
    cmp mjver,eax
    jge Tlib1  ; choose the greater value for version
    mov mjver,eax
Tlib1:
    inc idx
    jmp TlibLoop
   .endif ;RegEnumKeyEx
   invoke RegCloseKey,HKey
```

The number of major version is obtained; now it's possible to convert the string representation of GUID into the numeric one and pass it to LoadRegTypeLib function that accepts the following arguments:

• address of a sturcture holding GUID of the library being loaded;
• major version number (the exact value must be indicated; in this case it's in the mjver variable);
• minor version number (may be 0 here, because the system will load any library having minor version number greater than the indicated one);
• national language code of the library being loaded (lcid; may be neutral - 0);
• address of a variable that receives a pointer to ITypeLib interface on return.

```
  invoke MultiByteToWideChar,CP_ACP,0,ADDR buf,-1,ADDR wbuf,510
  invoke CLSIDFromString,ADDR wbuf,ADDR TlibGuid
  .if eax==NOERROR
   invoke LoadRegTypeLib,ADDR TlibGuid,mjver,0,0,ADDR pTlib
   .if eax==S_OK
```

The returned value S_OK means we've got a valid pointer to ITypeLib interface. We need GetDocumentation method; it accepts the following arguments:

• index of the type description; if -1, then the documentation for the library itself is returned (just that we need);
• address of a variable that receives a pointer to a BSTR string with the name of the requested element (in this case - of the type library itself);
• address of a variable that receives a pointer to a BSTR string with the description of the requested element;
• address of a variable that receives the Help context identifier;
• address of a variable that receives a pointer to a BSTR string that contains the fully qualified name of the Help file.

If you don't need any type of data you may pass 0 as the value of the proper argument. The address of GetDocumentation function is located in the virtual table of the ITypeLib interface with offset 24h.

```
  push OFFSET HelpFile
  push 0    ; don't need context, pBstrHelpCtx=NULL
  push OFFSET DocString
  push OFFSET CompName
  push -1    ; index (-1='the type library itself')
```

```
 mov eax,pTlib
 push eax   ; ‘this’ ptr
 mov eax,[eax]   ; virtual table
 call dword ptr [eax+24h] ; GetDocumentation
```

0 in HelpFile on return means there is no help file. In this case a simple message box with the received data is displayed:

```
.if HelpFile==0
 invoke WideCharToMultiByte,CP_ACP,0,CompName,-1,ADDR SubKeyBuf,255,0,0
 invoke lstrcpy,ADDR wbuf,ADDR SubKeyBuf
 invoke lstrcat,ADDR wbuf,ADDR CRLF
 invoke WideCharToMultiByte,CP_ACP,0,DocString,-1,ADDR SubKeyBuf,255,0,0
 invoke lstrcat,ADDR wbuf,ADDR SubKeyBuf
 invoke lstrcat,ADDR wbuf,ADDR CRLF
 invoke lstrcat,ADDR wbuf,ADDR ms1
 invoke MessageBox,0,ADDR wbuf,ADDR App,MB_OK or MB_ICONINFORMATION
```

But if the Help file is indicated the corresponding string is converted from Unicode into ANSI and its address is passed as an argument to ShellExecute function (in order to have possibility to load both hlp- and chm-files):

```
.else   ; load and show Help file
 invoke WideCharToMultiByte,CP_ACP,0,HelpFile,-1,ADDR buf,255,0,0
 invoke ShellExecute,0,ADDR cmd,ADDR buf,0,0,SW_SHOW
 .if eax<33   ; error of ShellExecute
  invoke wsprintf,ADDR wbuf,ADDR fmt,ADDR buf
  invoke MessageBox,0,ADDR wbuf,ADDR App,MB_OK or MB_ICONERROR
 .endif
.endif  ;HelpFile==0
```

BSTR strings must be freed aftewords to avoid memory leaks:

```
 invoke SysFreeString,HelpFile
 invoke SysFreeString,CompName
 invoke SysFreeString,DocString
```

On error of GetDocumentation method the proper message is displayed. In any case the ITypeLib interface pointer (pTlib) must be freed after that:

```
 .else    ; eroor invoking GetDocumentation
  invoke MessageBox,0,ADDr Err6,ADDR App,MB_OK or MB_ICONERROR
 .endif     ; GetDocumentation
 mov eax,pTlib
 push eax    ; ‘this’ ptr
 mov eax,[eax]    ; virtual table
 call dword ptr [eax+8]   ; function #3 (Release)
```

The function is concluded with handling of other errors:

```
  .else  ; error of LoadRegTypeLib
   invoke MessageBox,0,ADDR Err5,ADDR App,MB_OK or MB_ICONERROR
  .endif   ; LoadRegTypeLib==S_OK
 .else    ; error of CLSIDFromString
  invoke MessageBox,0,ADDR Err4,ADDR App,MB_OK or MB_ICONERROR
 .endif    ; CLSIDFromString==NOERROR
.else     ; error of RegOpenKeyEx
 invoke MessageBox,0,ADDR Err7,ADDR App,MB_OK or MB_ICONERROR
 ret
.endif   ; RegOpenKeyEx==ERROR_SUCCESS
ret
FindHelp endp
```

You can find out many interesting things using this utility. Try it!

AppID section

The registry key "HKEY_CLASSES_ROOT\AppID" jointly with the key "HKEY_LOCAL_MACHINE\Software\Microsoft\OLE" determine settings for the distributed COM (DCOM)

system.

The distributed COM system deals with data transmission via network and right away faces questions of authentication, authorization and other problems of data protection. The general configuration scheme is usually as follows: values of default security parameters are described under "HKEY_LOCAL_MACHINE\Software\Microsoft\OLE" key. The key "HKEY_CLASSES_ROOT\AppID" contains subkeys (GUID) that can store security parameters for separate groups of components (in this case "HKEY_CLASSES_ROOT\CLSID\{clsid of component}\AppID" containes a reference to the corresponding key under "HKEY_CLASSES_ROOT\AppID"). The values under AppID prevail over the respective default values. Besides the security parameters and other related to distributed system parameters may be indicated explicitly when creating component using the function CoCreateInstanceEx or like. In this case the explicitly specified values are used.

"HKEY_LOCAL_MACHINE\Software\Microsoft\OLE" key may have the following named values:

• EnableDCOM - permits (Y or y) or prohibits (N or n) remoute clients to launch components on this system;
• DefaultLaunchPermission - defines the default list of principals (ACL) who can launch components on this machine;
• DefaultAccessPermission – defines the default list of principals (ACL) who can access loaded objects;
• LegacyAuthenticationLevel – sets the default authentication level;
• LegacyImpersonationLevel – sets the default impersonation level;
• LegacyMutualAuthentication – permits (Y or y) or prohibits (any other value or the lack of this named value) mutual authentication;
• LegacySecureReferences – indicates whether invocations of AddRef and Release methods are secured (Y or y) or not (any other value or the lack of this named value).

The keys "HKEY_CLASSES_ROOT\AppID\{GUID}" can have the following named values:

• RemoteServerName – the name of the server where the component is to be launched;
• ActivateAtStorage – indicates that the component must be launched on the same machine as object data;
• LocalService – indicates that the component is implemented as Win32 service;
• ServiceParameters – is used in common with LocalService. The value of this parameter is passed on invocation of the proper service as parameters;
• RunAs - sets an application to run only as a given user;
• LaunchPermission - determins the list of principals (ACL) who can launch the application;
• AccessPermission - determins the list of principals (ACL) who can access objects of the given class;
• DllSurrogate - contains the full pathname to the wrapper application (exe) in which the remote inprocess (dll) server is to be activated;
• AuthenticationLevel - sets the authentication level for AppID.

The indicated in the registry values may be overlapped through the explicit call to CoInitializeSecurity function.

Sorry for my English; I hope you did understand it.

3 кита COM. Кит второй: dll

Автор: Roustem

Дата: 2007-01-03

Скачать материалы для статьи (находится в архиве `wasm_files.zip`*:* `files/0446/comkit2.rar`*)*

Второй важнейшей основой, без которой невозможна инфраструктура COM, является использование DLL. Причем не просто использование, а очень активное использование; и использование не только самих DLL, но и тех принципов, которые применяются в технологии динамического связывания.

Технология COM претендует на универсальность своей компонентной модели. Это значит, что безболезненно взаимодействовать между собой должны не только компоненты, написанные на разных языках и созданные с использованием различных сред разработки, но и разные виды компонентов: и созданный специально для внедрения в другие приложения код DLL; и компоненты, входящие в EXE-модули самостоятельных приложений; и код, размещенный на другой машине и потенциально созданный для другой платформы. Естественно, сразу возникает проблема передачи данных через границы процессов и систем, а также единообразия работы с внутрипроцессными и внепроцессными объектами.

COM в качестве решения этой проблемы использует единый и единственный указатель интерфейса: все, что нужно для работы с объектом - это получить указатель на его интерфейс, и ничего более. Этот указатель представляет собой некий адрес в адресном пространстве клиента, по которому соответствующая область памяти структурирована определенным образом в соответствии с бинарным стандартом. А это значит, что область эта может принадлежать только DLL, и любой объект - будь он в другом процессе или на другой машине - имеет свою DLL (неважно, созданную разработчиком компонента или предоставленную системой), которая внедряется в адресное пространство клиента. Более того, именно эта часть объекта и предоставляет клиенту собственно интерфейс; можно сказать, что интерфейс объекта всегда реализуется в DLL, независимо от того, внутрипроцессный это объект, внепроцессный или удаленный.

"Интерфейсная" область структурируется иерархически с использованием других указателей наподобие связанного списка. Начинающих - и даже не только начинающих - COM-программистов часто смущает это обилие вложенных указателей. Чего стоит, например, аргумент метода QueryInterface в виде двойного указателя на неопределенный тип! Один этот void языка C++ способен свести с ума, потому что понять его смысл невозможно в принципе. А если кто-то попытался использовать C и выяснил, что там на самом деле есть еще и скрываемая объектными языками виртуальная таблица - чтобы добраться до метода, необходимо тройное перенаправление?..

Даже авторы некоторых статей по COM-программированию на ассемблере как-то слишком уж поспешно проскакивают этот момент, бросаясь сразу в объятия спасительных макросов и предопределенных структур и пробормотав что-то невнятное о запутанности указателей, так что возникает подозрение, что они сами не до конца разобрались с этой проблемой. Между тем это вопрос, в котором можно (и нужно) разобраться - один раз, но досконально.

Попробуем спроектировать собственную объектную модель и посмотрим, какие проблемы при этом возникают. Главной идеей ООП было объединить данные и обрабатывающие их и семантически связанные с ними функции в одну конструкцию. Предположим, что у нас есть указатель на некий объект, представляющий собой область с данными, за которой непосредственно следуют обрабатывающие их методы (рис. 1). Доступ и к данным, и к методам этого объекта можно

получить, используя простое смещение, добавляемое к значению указателя - как при работе с обычной структурой. Например, если указатель на объект находится в регистре eax, метод 2 можно вызвать следующим способом:

```
add eax,m2
call eax
```

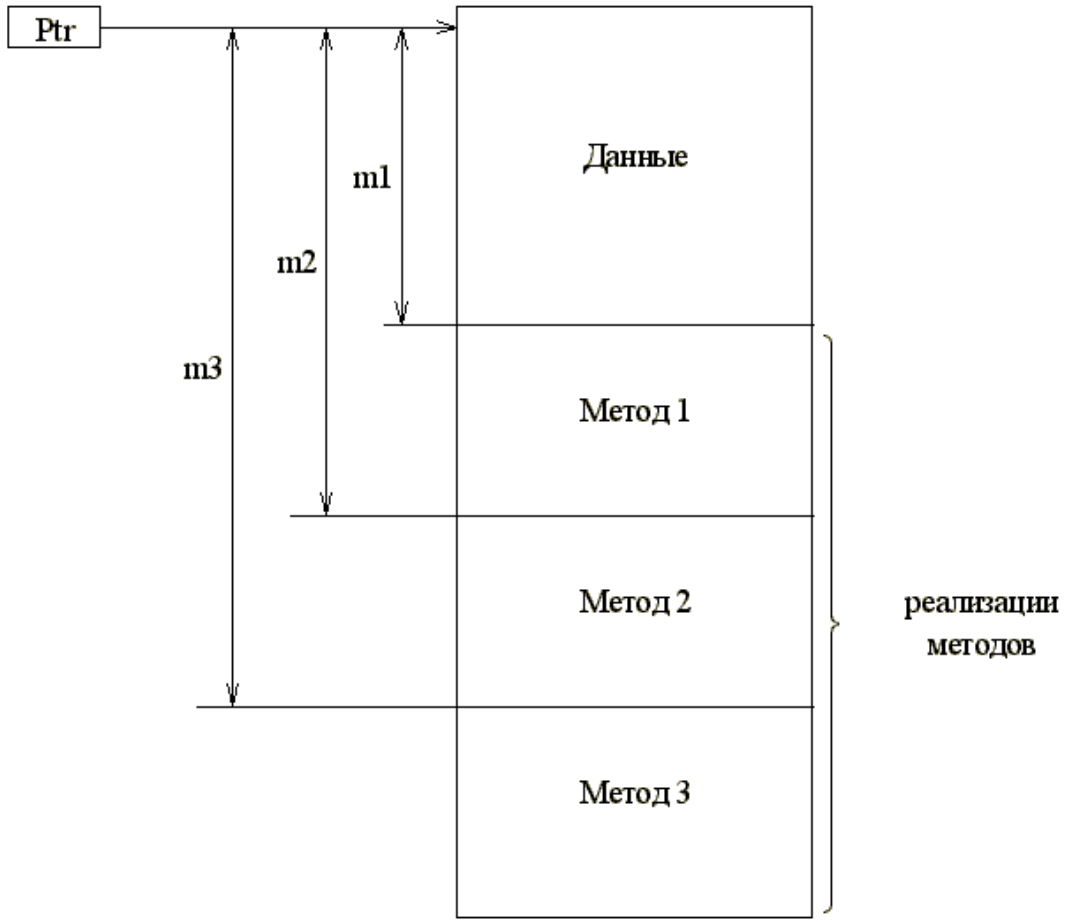


Рис. 1. Гипотетическая модель объекта, доступ к методам и данным которого осуществляется через один и тот же указатель

Какие могут тут быть проблемы? Если создаются несколько экземпляров одного и того же объекта, в памяти оказываются несколько копий одного и того же кода, хотя это совершенно излишне. Для решения этой проблемы данные и код можно разместить в различных секциях, причем секцию кода сделать общей, а секцию данных для каждого экземпляра объекта выделить свою. Именно по такому принципу строятся DLL (хотя формат PE-файла и соответствующие опции компиляторов позволяют создавать при необходимости и общие секции данных).

Как только появляется общая секция кода, возникает и вопрос отслеживания количества ссылок на него, чтобы своевременно освобождать память после завершения работы. Обычно эта проблема решается созданием счетчиков, значение которых увеличивается на один при создании нового экземпляра и уменьшается на один при его разрушении. Когда значение счетчика при очередном уменьшении достигает 0, секцию кода тоже можно выгрузить из памяти. Этот механизм используется операционной системой при загрузке и выгрузке DLL, но он не виден приложениям; в компонентной же модели он становится явным.

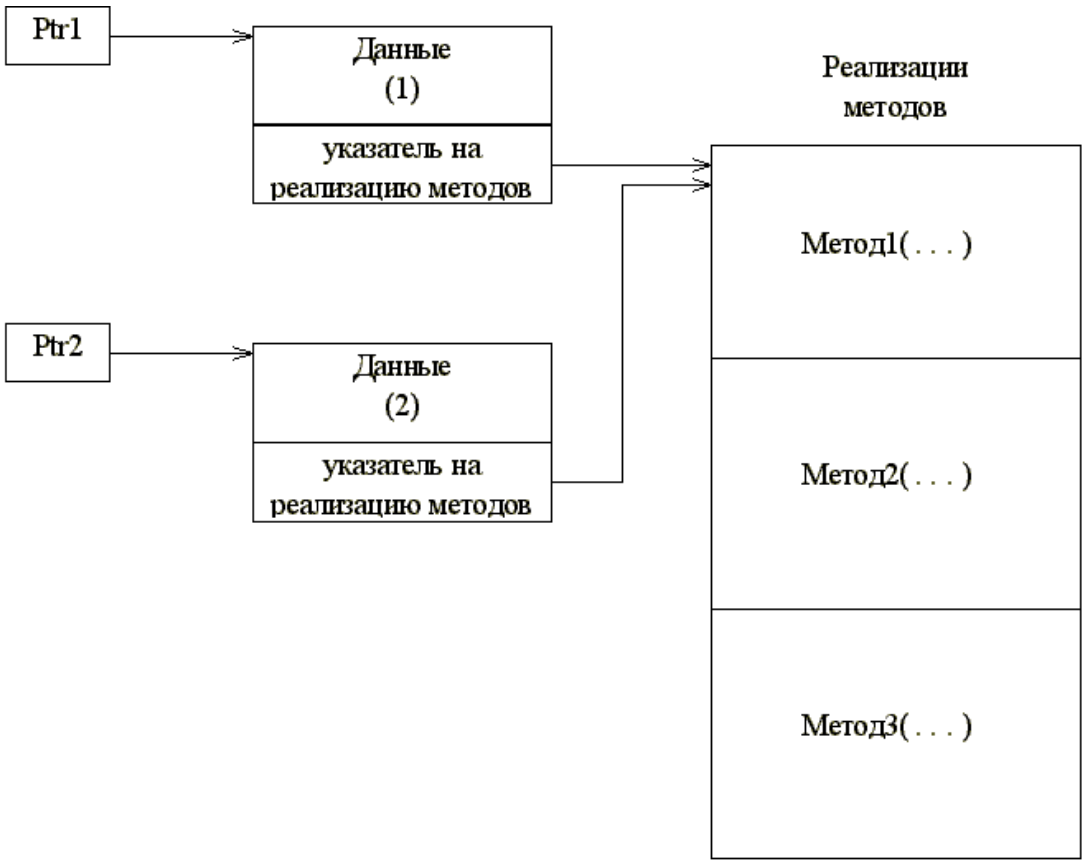


Рис. 2. Модель с различными секциями данных и единой секцией кода.

Для того, чтобы обращаться и к данным, и к коду через один и тот же указатель объекта, в области данных можно записать указатель на общую для всех экземпляров секцию кода с реализацией методов объекта (рис. 2). При такой схеме методы объекта не имеют непосредственного неявного доступа к данным и последние придется передавать через аргументы, как в случае самых обычных процедур. А это разрушает саму идею объектного подхода - стоило ли ради этого огород городить?

Поэтому возникает еще одна схема, реализующая один из основных принципов ООП - инкапсуляцию: непосредственный доступ к данным объекта через указатель на объект закрывается - он превращается в "чёрный ящик" (рис. 3). Из всех данных доступным остается лишь указатель на секцию кода (благодаря тому, что он расположен в самом начале структуры данных и именно его адрес содержит указатель объекта) с реализацией методов объекта, которые непосредственно имеют дело с данными. Поскольку у каждого объекта своя секция данных, тогда как реализация методов одна для всех объектов, каждый метод при своем вызове в качестве первого параметра получает указатель на секцию данных именно того объекта, для которого вызывается метод. (В терминологии языка C++ этот параметр получил название указателя "this").

"Расплатой" за это является один уровень перенаправления, добавляемый на пути от указателя объекта к его методу. Теперь, если указатель объекта находится в регистре eax, метод вызывается следующим образом:

```
   ; перенаправление:
mov eax,[eax] ; получить адрес секции кода
add eax,m2 ; добавить смещение второго метода
call eax
```

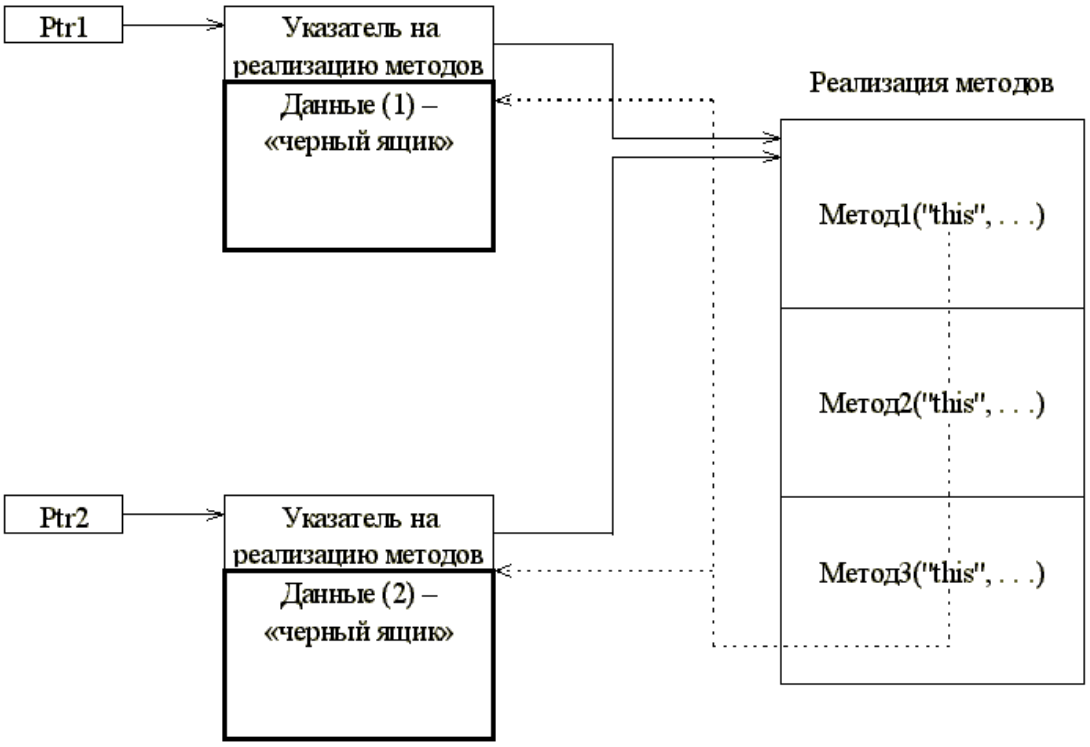


Рис. 3. Реализация принципа инкапсуляции.

Еще одна проблема возникает в связи с реализацией другого основного принципа ООП - полиморфизма. В среде COM времени исполнения нет имен. В принципе можно было бы создать и систему с использованием имен во время исполнения - например, DLL пользуются именно такой схемой. (К слову сказать, основанный на COM большой раздел OLE - автоматизация - использует модель позднего связывания, построенную именно по такому принципу. Но об этом поговорим как-нибудь в другой раз). Для доступа к методам вычисляются их адреса с использованием указателя интерфейса и постоянных смещений, а имена могут существовать лишь во время компиляции.

Традиционные языки программирования и среды разработки используют подобную же схему, однако в их случае вычисляемые смещения уникальны для каждой конкретной компиляции и они не используются нигде, кроме данного приложения. В случае же с компонентной моделью один и тот же интерфейс могут реализовать совершенно различные компоненты, и тут проявляется проблема смещений.

Предположим, имеются две реализации одного и того же интерфейса, содержащего три метода. Как видно на рис. 4, в двух реализациях методы требуют разного количества памяти, поэтому их смещения относительно начала секции кода оказываются различными. В результате код, который может работать с первой реализацией, не может работать со второй, и наоборот - полиморфизм невозможен.

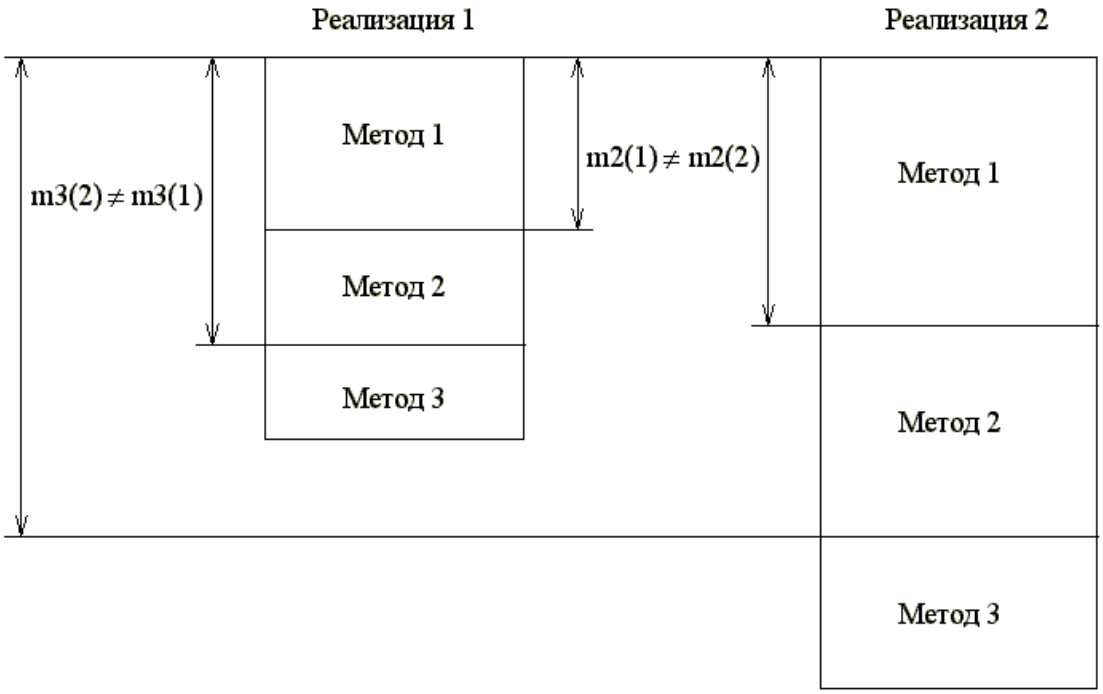


Рис. 4. Проблема различных реализаций одного интерфейса.

Для решения этой проблемы вводится еще один посредник между указателем на объект и реализацией его методов - виртуальная таблица (рис. 5). В секции данных объекта теперь располагается указатель не на секцию кода, а на виртуальную таблицу, в которой содержатся указатели на соответствующие методы. Смещение указателя на один и тот же метод относительно начала виртуальной таблицы всегда постоянно, хотя объем кода метода и его расположение в секции кода в различных реализациях могут сильно различаться. Но это вводит еще один уровень перенаправления. Вызов метода выглядит теперь следующим образом:

```
   ; первое перенаправление:
mov eax,[eax]  ; получить адрес виртуальной таблицы
   ; второе перенаправление:
mov eax,[eax+m2] ; получить адрес метода по смещению
   ; в виртуальной таблице
call eax
```

Правда, это может быть записано короче таким образом:

```
mov eax,[eax]  ; первое перенаправление
call dword ptr [eax+m2] ; косвенный вызов процедуры
   ; со вторым перенаправлением
```

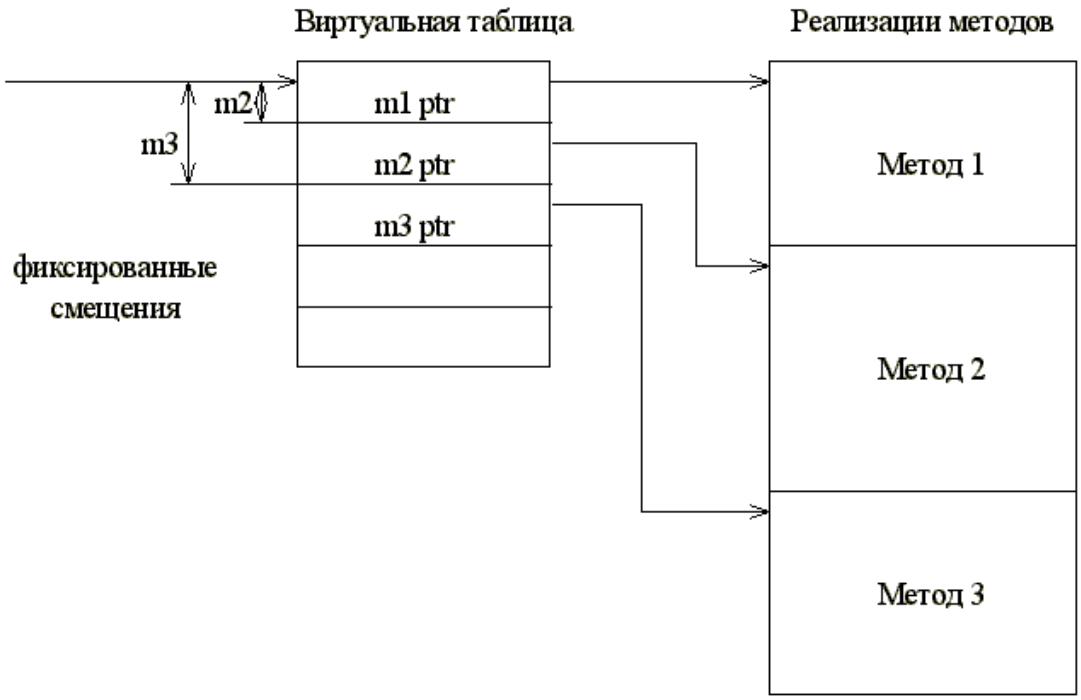


Рис. 5. Реализация принципа полиморфизма.

Полиморфизм означает, что в качестве реализации метода интерфейса может выступать абсолютно любой код, удовлетворяющий данному синтаксису вызова метода. Именно это и используется в реализации знаменитой "прозрачности удаленного доступа" COM (хотя этот подход использовался еще до COM в RPC). Благодаря полиморфизму код, реализующий интерфейс в той части сервера, которая размещается в DLL и загружается в адресное пространство клиента, может исполнять не алгоритм метода, а заниматься совершенно другими вещами - упаковывать полученные для метода параметры со всеми сопутствующими данными в некую структуру и отправлять ее куда-то в адресное пространство другого процесса или даже по сети на другую машину, где находится другая часть кода, реализующая собственно алгоритм метода. DLL выступает в этом случае "представителем" кода, размещенного в другом процессе или на другой системе. Клиент же ничего этого не видит; максимум, что он может заметить - увеличение задержек при выполнении методов.

Если свести все воедино, мы получим схему, приведенную на рис. 6. Это и является бинарной структурой интерфейса, используемой в COM. Таким образом, несколько упрощая, можно сказать, что первое перенаправление указателя интерфейса COM на пути к реализации метода делается "во славу" и ради принципа инкапсуляции, а второе - "во славу" и ради принципа полиморфизма. Возможно, это поможет кому-нибудь лучше запомнить эту схему и не запутаться в ней. :)

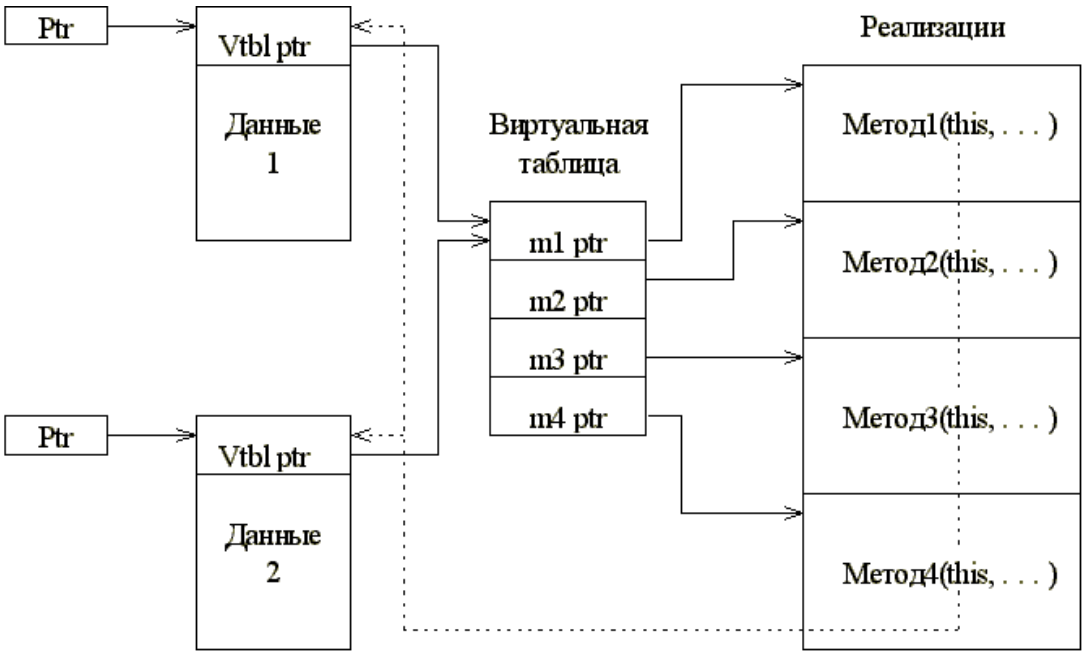


Рис. 6. Бинарная структура интерфейса COM.

Осталось лишь выяснить, откуда берутся двойные указатели на интерфейс и зачем нужно третье перенаправление. На самом деле, это искусственное абстрактное образование, создаваемое языками высокого уровня и не имеющее собственно к COM никакого отношения. Дело в том, что указатель интерфейса должен быть возвращен функцией, создающей объект. Но функции COM по соглашению возвращают результат типа HRESULT, сообщающий об успешности или ошибке хода выполнения. Для возврата указателя используется аргумент функции, а это значит, что указатель передается по ссылке: его значение копируется в переменную, которую подготовил клиент и передал функции ее адрес. Формально тип соответствующего аргумента функции оказывается двойным указателем; но гораздо проще понимать его как адрес переменной (на стороне клиента), в которую должен быть возвращен указатель интерфейса. И соответственно, следует не создавать переменную типа двойного указателя (в нотации C - void ppv) и передавать функции ее значение (ppv), а создать переменную типа указателя на интерфейс (void* pv) и передать функции ее адрес (&pv).

Раз уж речь шла о принципах ООП, придется упомянуть еще и третий основной принцип - наследования. На уровне интерфейсов COM он проявляется в том, что интерфейс-потомок наследует всю виртуальную таблицу интерфейса-родителя, добавляя указатели к собственным методам в ее конец. Поскольку все интерфейсы COM прямо или косвенно наследуются от IUnknown, мы можем с абсолютной уверенностью заявить, что первые три указателя виртуальной таблицы любого интерфейса COM содержат адреса реализаций трех методов IUnknown в одном и том же порядке: QueryInterface, AddRef и Release (со смещениями относительно начала виртуальной таблицы соответственно 0, 4 и 8 байт). Т.е. если у нас есть указатель некоторого интерфейса pSomeInterface, эти методы вызываются так:

```
; здесь аргументы (если есть) помещаются в стек
mov eax,pSomeInterface
push eax   ; первый аргумент всех методов - "this"
mov eax,[eax]
call dword ptr [eax+x]  ; x=0 для QueryInterface,
    ; x=4 для AddRef,
    ; x=8 для Release
```

Обратите внимание, здесь речь идет о наследовании интерфейсов - т.е. абстрактных структур - а не реализаций, как в объектно-ориентированных языках программирования. Для наследования реализаций или, как это называется в COM, повторного использования кода, существует другой,

специально для этого созданный механизм, но это уже тема отдельной большой статьи.

Структура dll-сервера

Ну что ж, пора приступить к практике и посмотреть, каким образом задействуется та самая бинарная структура интерфейса COM, которую мы так долго разбирали. Система предоставляет своего рода "точки входа в COM". Клиентскую сторону мы уже видели - это функция API CoCreateInstance, с помощью которой мы создавали универсальный клиент в прошлой статье. Со стороны же сервера DLL это функция DllGetClassObject, которая должна быть реализована и экспортирована DLL. Система передает ей такие параметры:

• адрес структуры, содержащей CLSID объекта класса;
• адрес структуры, содержащей IID интерфейса, который запрашивается у объекта класса;
• адрес переменной, в которой возвращается указатель на затребованный интерфейс.

Эта вызываемая системой функция создает объект класса - но это не запрашиваемый клиентом объект. Да, вы узнали страшную тайну. COM - это царство посредников. Объект класса - это еще один посредник, имеющий даже собственное имя - "фабрика классов". Именно эта фабрика создает экземпляры объектов компонента для клиента, предоставляя для этой цели собственный интерфейс - чаще всего это IClassFactory, хотя могут быть и другие варианты.

Функция CoCreateInstance, которую мы использовали в нашем клиенте (COM2.exe (находится в архиве `wasm_files.zip`: `files/0446/comkit1.rar`)), на самом деле является "оберткой" вокруг функции, которая является действительной точкой входа со стороны клиента - CoGetClassObject. Параметры этой функции:

• адрес структуры, содержащей CLSID компонента, для которого создается фабрика классов;
• тип сервера (внутрипроцессный, локальный, удаленный или внутрипроцессный обработчик);
• адрес структуры, содержащей имя и другие параметры удаленного сервера (этот параметр может быть 0);
• адрес структуры, содержащей IID интерфейса фабрики классов (обычно IClassFactory);
• адрес переменной, в которой будет возвращен указатель на требуемый интерфейс.

CoCreateInstance вызывает CoGetClassObject, запрашивая у нее указатель на интерфейс IClassFactory для компонента с данным CLSID. Затем через полученный указатель вызывает метод CreateInstance интерфейса IClassFactory, запрашивая уже указатель на интерфейс самого компонента, а указатель на IClassFactory сразу же освобождает (вызвав через него метод Release).

Но об этом позже; а сейчас построим простейший сервер, вернее, псевдосервер COM - потому что он экспортирует DllGetClassObject, но функция эта все время возвращает код ошибки CLASS_E_CLASSNOTAVAILABLE. Сервер DLL должен экспортировать еще одну функцию - DllCanUnloadNow, которая в зависимости от состояния внутренних счетчиков реализуемых объектов возвращает S_OK или S_FALSE, в соответствии с чем система может выгрузить DLL из памяти. Вот исходный код полностью (файл COM_6.asm):

```
.386
.model flat,stdcall
option casemap:none

include \masm32\include\windows.inc
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib

.data
ms1 db "DllGetClassObject",0
ms2 db "DllCanUnloadNow",0
app db "FooDll",0
.code

DllGetClassObject proc rclsid:DWORD, riid:DWORD, ppv:DWORD
 invoke MessageBox,0,addr ms1,addr app,0
```

```
 mov eax,CLASS_E_CLASSNOTAVAILABLE
 ret
DllGetClassObject endp

DllCanUnloadNow proc
 invoke MessageBox,0,addr ms2,addr app,0
 mov eax,S_OK
 ret
DllCanUnloadNow endp
end
```

Необходимо создать еще def-файл (COM_6.def):

```
LIBRARY COM_6
EXPORTS DllGetClassObject PRIVATE
 DllCanUnloadNow PRIVATE
```

Ключевое слово "PRIVATE" используется для того, чтобы экспортируемые функции не попали в создаваемый при построении DLL lib-файл - чтобы кто попало не мог импортировать эти функции, ибо они предназначены только "для своих" (т.е. системы COM). Хотя это обстоятельство может остановить разве что делающих все по правилам прикладных программистов, но никак не низкоуровневиков, способных подключиться непосредственно к DLL. :)

Строим DLL со следующими опциями (обратите внимание, наша DLL в данном конкретном случае не имеет функции инициализации и соответственно точки входа):

```
\masm32\bin\ml /c /coff /Cp COM_6.asm
\masm32\bin\Link /DLL /DEF:COM_6.def /NOENTRY /SUBSYSTEM:WINDOWS /VERSION:4.0 /LIBPATH:\masm32\lib
    COM_6.obj
```

Предполагается, что MASM32 расположен в каталоге \MASM32 текущего диска; если это не так, необходимо соответствующим образом отредактировать пути.

Теперь запускаем редактор реестра и находим созданный в прошлый раз "хулиганский" ключ {00000000-0000-0000-0000-000000000001} в разделе "HKEY_CLASSES_ROOT\CLSID". Удалим оттуда подключ "TreatAs", если он там есть, и создадим новый: "InprocServer32". В качестве значения запишем полный путь к созданной нами DLL (например, "C:\COM_6\COM_6.dll"). Запускаем наш универсальный клиент (COM_2.exe), заполняем CLSID, для IID жмем кнопку "IUnknown", тип сервера - "InProc server" и - "Connect". Остальное не описываю - сами увидите. Надеюсь, то, что клиент сообщает об ошибке, никого не удивляет. :)

IClassFactory

Интерфейс IClassFactory, помимо функций-членов IUnknown, содержит два собственных: CreateInstance и LockServer. Приступим к реализации второй версии нашего "сервера" (файл COM_7.asm):

```
.386
.model flat,stdcall
option casemap:none

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\ole32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\ole32.lib

.data
pVtbl DWORD offset Vtbl
counter DWORD 0
```

Эти две переменные и составляют содержание простейшего инстанциированного "экземпляра интерфейса" - указатель на виртуальную таблицу (в соответствии с бинарным стандартом COM) и

счетчик ссылок для данного объекта. Других данных у нашей реализации этого объекта нет. Дальше расположим саму виртуальную таблицу:

```
Vtbl DWORD offset IUnknown_QueryInterface
 DWORD offset IUnknown_AddRef
 DWORD offset IUnknown_Release
 DWORD offset IClassFactory_CreateInstance
 DWORD offset IClassFactory_LockServer
```

На самом деле названия функций не играют никакой роли - можно было использовать любые названия. Важен порядок расположения функций в виртуальной таблице - он должен строго соответствовать интерфейсу. Само собой, указатели на функции нельзя чередовать с посторонними данными. В данном случае, в названии первых трех функций имеется префикс IUnknown, чтобы подчеркнуть, что эти члены наследуются от базового интерфейса, хотя все эти функции являются членами интерфейса IClassFactory.

Далее следуют обычные данные:

```
IUnk dd 0 ; IID интерфейса IUnknown
 dw 0
 dw 0
 db 0C0h,0,0,0,0,0,0,46h
ICFct dd 1 ; IID интерфейса IClassFactory
 dw 0
 dw 0
 db 0C0h,0,0,0,0,0,0,46h
ms db "Unloading",0
ms0 db "DllGetClassObject",0
ms1 db "QueryInterface",0
ms2 db "AddRef",10,13,"counter = %i",0
ms3 db "Release",10,13,"counter = %i",0
ms4 db "CreateInstance",0
ms5 db "LockServer",10,13,"counter = %i",0
app db "ClassFactory",0
buf db 128 dup(0)
```

Теперь реализации функций и методов.

```
.code
DllGetClassObject proc rclsid:DWORD, riid:DWORD, ppv:DWORD
 invoke MessageBox,0,offset ms0,offset app,0
 ; перенаправить вызов IUnknown_QueryInterface
 push ppv ; ppvObject
 push riid ; iid
 lea eax,pVtbl
 push eax ; указатель 'this'
 call IUnknown_QueryInterface
 ret  ; возвратить результат QueryInterface
DllGetClassObject endp
```

В данной реализации эта функция (как и остальные) отображает окно сообщения, а затем просто перенаправляет вызов функции интерфейса IUnknown_QueryInterface. Параметр rclsid нужен в тех случаях, когда один сервер dll реализует сразу несколько компонентов с различными CLSID (и функция DllGetClassObject должна решить, объект какого именно класса создавать). В нашем случае "компонент" всего один - вернее, его нет вообще, поэтому этот "сервер" будет работать с любым CLSID. В случае реального сервера именно в этом месте CLSID реализуемых компонентов жестко вшиты в код.

Обратите внимание, что мы вызываем IUnknown_QueryInterface напрямую. Хотя это и метод интерфейса COM, но мы, как авторы реализации, обладаем знанием внутреннего устройства нашего объекта, которого нет у клиента. При желании ничто не может нам помешать вызвать этот метод стандартным для COM способом, т.е. через указатель интерфейса - в данном случае, например, используя регистр eax, хотя это не имеет смысла.

Поскольку теперь у нас есть счетчик объекта, DllCanUnloadNow должна возвращать результат, зависящий от его значения:

```
DllCanUnloadNow proc
 .if counter>0
  mov eax,S_FALSE
 .else
  invoke MessageBox,0,offset ms,offset app,0
  mov eax,S_OK
 .endif
 ret
DllCanUnloadNow endp
```

Переходим к самому интересному - реализации QueryInterface:

```
IUnknown_QueryInterface proc thisptr:DWORD,iid:DWORD,ppvObject:DWORD
 invoke MessageBox,0,offset ms1,offset app,0
 invoke IsEqualGUID,offset IUnk,iid
 .if eax==0  ; не IUnknown
  invoke IsEqualGUID,offset ICFct,iid
  .if eax==0 ; не IClassFactory
   mov eax,E_NOINTERFACE
   ret
  .endif
 .endif
```

Сначала с помощью вспомогательной функции IsEqualGUID (принимающей в качестве аргументов адреса двух содержащих GUID'ы структур, которые необходимо сравнить между собой) проверяем, является ли запрошенный интерфейс IUnknown или IClassFactory. Если ни то, ни другое, возвращается ошибка. Если же запрошен один из этих двух интерфейсов, в переменную, адрес которой передан в аргументе ppvObject, копируется адрес переменной pVtbl, находящейся в самом начале нашего виртуального "объекта". Поскольку функция возвращает новый указатель на объект, должен быть увеличен его счетчик, что также делается непосредственным вызовом IUnknown_AddRef:

```
 mov eax,ppvObject ; адрес переменной для результата
 lea ecx,pVtbl  ; указатель на IClassFactory
 mov [eax],ecx
 push ecx  ; указатель 'this'
 call IUnknown_AddRef
 mov eax,S_OK
 ret
IUnknown_QueryInterface endp
```

Реализации AddRef и Release тривиальны - они просто соответственно увеличивают или уменьшают значение счетчика объекта и отображают окно сообщения с его текущим содержимым. Обычно при достижении счетчиком 0 реализация Release должна удалить объект, освобождая занимаемую им память. В нашем же случае никакой объект реально не создавался, все данные были статическими, поэтому и уничтожать ничего не надо.

```
IUnknown_AddRef proc thisptr:DWORD
 inc counter
 invoke wsprintf,offset buf,offset ms2,counter
 invoke MessageBox,0,offset buf,offset app,0
 mov eax,counter
 ret
IUnknown_AddRef endp

IUnknown_Release proc thisptr:DWORD
 dec counter
 invoke wsprintf,offset buf,offset ms3,counter
 invoke MessageBox,0,offset buf,offset app,0
 mov eax,counter
 ret
IUnknown_Release endp
```

Теперь собственные функции интерфейса IClassFactory. CreateInstance принимает, помимо указателя 'this', еще 3 аргумента:

• указатель на "внешний" IUnknown, который применяется при агрегировании объектов. Если агрегирования нет, этот аргумент равен NULL;
• адрес структуры, содержащей IID запрошенного интерфейса;
• адрес переменной, в которую должен быть возвращен указатель интерфейса.

Поскольку наш "сервер" не реализует ни одного компонента, он просто на все запросы возвращает E_NOINTERFACE:

```
IClassFactory_CreateInstance proc thisptr:DWORD,pUnkOuter:DWORD,riid:DWORD,ppvObject:DWORD
 mov eax,ppvObject
 xor ecx,ecx
 mov [eax],ecx ; указатель интерфейса = NULL
 invoke MessageBox,0,offset ms4,offset app,0
 mov eax,E_NOINTERFACE
 ret
IClassFactory_CreateInstance endp
```

Метод LockServer кроме 'this' принимает лишь один аргумент - булевую переменную, указывающую, увеличить (TRUE) или уменьшить (FALSE) значение счетчика замка сервера. Замок позволяет сохранить сервер в памяти (невыгруженным), даже когда счетчики всех объектов равны 0. В нашем случае замок использует общую со счетчиком объекта глобальную переменную:

```
IClassFactory_LockServer proc thisptr:DWORD,fLock:DWORD
 .if fLock
  inc counter
 .else
  dec counter
 .endif
 invoke wsprintf,offset buf,offset ms5,counter
 invoke MessageBox,0,offset buf,offset app,0
 mov eax,S_OK
 ret
IClassFactory_LockServer endp
end
```

Вот и вся программа. Def-файл такой же, как в предыдущем случае (COM_7.def):

```
LIBRARY COM_7
EXPORTS DllGetClassObject PRIVATE
  DllCanUnloadNow PRIVATE
```

Это одна из возможных реализаций "точки входа" COM и фабрики классов, притом не самая оптимальная (зато простая). Реализация сильно зависит в том числе и от применяемой модели многопоточности COM (это тема для отдельной большой статьи). В нашем случае как сервер, так и его клиент однопоточные - система COM будет осуществлять принудительную синхронизацию поступающих серверу запросов, что в некоторых случаях может сильно снижать производительность. Некоторые свидетельства "посредничества" COM можно пронаблюдать, экспериментируя с данной утилитой; в частности, попробуйте повторный вход (re-entrance) в сервер из клиента, нажав на кнопку "Connect" (с теми же параметрами) в середине процесса обработки первого запроса (при отображении окна сообщения с AddRef), и сравните это с результатами, когда запущены одновременно два (или более) клиента для одного и того же сервера.

Реализация простейшего объекта

Добавим к нашему серверу "компонент". Для простоты он будет содержать единственный интерфейс IUnknown, но создавать его экземпляры мы будем с выделением памяти из кучи. Благодаря этому мы сможем, наконец, увидеть настоящее применение указателя "this" (даже в таких простейших, казалось бы, методах как AddRef и Release).

Каждый экземпляр объекта будет представлен небольшой структурой, первый член которой (в соответствии со спецификацией COM) является указателем на виртуальную таблицу реализации методов IUnknown. У нас уже есть реализация IUnknown для объекта класса, но мы должны написать другую реализацию методов этого же интерфейса, поскольку объект компонента и объект класса - разные объекты (вот вам принцип полиморфизма в действии). Вторым элементом в структуре объекта компонента будет счетчик ссылок для данного экземпляра объекта, а третьим - идентификатор объекта (для того, чтобы отображать его в окне сообщения). В качестве идентификатора используем простой счетчик созданных объектов, который будет вести объект фабрики классов; при создании это значение будет скопировано в соответствующее поле структуры объекта.

Для представления экземпляра объекта удобнее всего было бы использовать структуру MASM, но в данной реализации я использовал непосредственное обращение к соответствующим данным на низком уровне, чтобы как следует "прочувствовать", что же это такое. Желающие могут переписать этот пример, используя высокоуровневые конструкции MASM'a - весьма неплохое упражнение для закрепления материала.

Перейдем к коду (COM_8.asm):

```
.386
.model flat,stdcall
option casemap:none

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\ole32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\ole32.lib

.data
objcnt DWORD 0 ;тот самый счетчик созданных объектов
```

Статическая структура для "объекта класса":

```
pVtbl  DWORD offset Vtbl
counter DWORD 0
```

За ней следуют две виртуальные таблицы: первая - для реализации интерфейса IClassFactory объекта класса, вторая - для реализации (единственного) интерфейса IUnknown объекта компонента:

```
; виртуальная таблица для объекта класса
Vtbl DWORD offset IClassFactory_QueryInterface
 DWORD offset IClassFactory_AddRef
 DWORD offset IClassFactory_Release
 DWORD offset IClassFactory_CreateInstance
 DWORD offset IClassFactory_LockServer

; виртуальная таблица для объекта компонента
Vtbl1 DWORD offset IUnknown_QueryInterface
 DWORD offset IUnknown_AddRef
 DWORD offset IUnknown_Release
```

Далее идут обычные данные:

```
IUnk dd 0 ; IID интерфейса IUnknown
 dw 0
 dw 0
 db 0C0h,0,0,0,0,0,0,46h
ICFct dd 1 ; IID интерфейса IClassFactory
 dw 0
 dw 0
 db 0C0h,0,0,0,0,0,0,46h
ms db "Unloading",0
```

```
ms0 db "DllGetClassObject",0
ms1 db "ClassFactroy:",10,13,"QueryInterface",0
ms2 db "ClassFactory:",10,13,"AddRef",10,13,"counter = %i",0
ms3 db "ClassFactory:",10,13,"Release",10,13,"counter = %i",0
ms4 db "ClassFactory:",10,13,"CreateInstance",0
ms5 db "ClassFactory:",10,13,"LockServer",10,13,"counter = %i",0
app db "DllServer",0
ms6 db "Object %i:",10,13,"QueryInterface",0
ms7 db "Object %i:",10,13,"AddRef",10,13,"objref: %i",0
ms8 db "Object %i:",10,13,"Release",10,13,"objref: %i",0
buf db 128 dup(0)
```

Реализация "точек входа" и интерфейса IUnknown объекта класса практически не претерпела никаких изменений, за исключением того, что в DllCanUnloadNow было добавлено условие проверки значения счетчика созданных объектов objcnt.

```
.code

DllGetClassObject proc rclsid:DWORD, riid:DWORD, ppv:DWORD
 invoke MessageBox,0,offset ms0,offset app,0
 ; перенаправить вызов IClassFactory_QueryInterface
 push ppv ; ppvObject
 push riid ; iid
 lea eax,pVtbl
 push eax ; указатель 'this'
 call IClassFactory_QueryInterface
 ret  ; возвращает результат вызова QueryInterface
DllGetClassObject endp

DllCanUnloadNow proc
 .if counter>0 || objcnt>0
  mov eax,S_FALSE
 .else
  invoke MessageBox,0,offset ms,offset app,0
  mov eax,S_OK
 .endif
 ret
DllCanUnloadNow endp

IClassFactory_QueryInterface proc thisptr:DWORD,iid:DWORD,ppvObject:DWORD
 invoke MessageBox,0,offset ms1,offset app,0
 .if ppvObject==0
  mov eax, E_INVALIDARG
  ret
 .endif
 invoke IsEqualGUID,offset IUnk,iid
 .if eax==0  ; не  IUnknown
  invoke IsEqualGUID,offset ICFct,iid
  .if eax==0 ; не IClassFactory
   mov eax,E_NOINTERFACE
   ret
  .endif
 .endif
 ; iid является либо IUnknown, либо IClassFactory
 mov eax,ppvObject ; адрес переменной для указателя интерфейса
 lea ecx,pVtbl  ; указатель на IClassFactory
 mov [eax],ecx
 push ecx  ; указатель 'this'
 call IClassFactory_AddRef
 mov eax,S_OK
 ret
IClassFactory_QueryInterface endp

IClassFactory_AddRef proc thisptr:DWORD
 inc counter
 invoke wsprintf,offset buf,offset ms2,counter
 invoke MessageBox,0,offset buf,offset app,0
 mov eax,counter
 ret
IClassFactory_AddRef endp
IClassFactory_Release proc thisptr:DWORD
```

```
 dec counter
 invoke wsprintf,offset buf,offset ms3,counter
 invoke MessageBox,0,offset buf,offset app,0
 mov eax,counter
 ret
IClassFactory_Release endp
```

Зато существенные изменения претерпел метод CreateInstance. Теперь он создает настоящие объекты, выделяя для них память из кучи. Кроме того, нужно организовать проверку параметров и соответствующую обработку ошибок:

```
IClassFactory_CreateInstance proc uses ebx thisptr:DWORD,pUnkOuter:DWORD,
riid:DWORD,ppvObject:DWORD
 invoke MessageBox,0,offset ms4,offset app,0
 .if ppvObject==0
  mov eax,E_INVALIDARG
  ret
 .endif
 mov ebx,ppvObject ; сохраним в регистре адрес переменной
    ; для указателя интерфейса объекта
.if pUnkOuter!=0
 xor ecx,ecx
 mov [ebx],ecx
  mov eax,CLASS_E_NOAGGREGATION
  ret
 .endif
```

Если кто-то пытается агрегировать наш объект в состав какого-то другого объекта, необходимо сообщить ему, что мы на это вовсе не рассчитывали. Параметр pUnkOuter должен содержать NULL. Необходимо проверить также параметр riid: наш компонент поддерживает единственный интерфейс - IUnknown.

```
 invoke IsEqualGUID,offset IUnk,riid
 .if eax==0 ; не IUnknown
  mov [ebx],eax
  mov eax,E_NOINTERFACE
  ret
 .endif
```

Теперь "сердцевина" метода: создаем объект. Как уже упоминалось, он представлен структурой из трех полей размером DWORD каждое, поэтому необходимо выделить из кучи блок памяти в 12 байт:

```
 invoke LocalAlloc,LPTR,12
 .if eax==0 ; ошибка выделения памяти
  mov [ebx],eax
  mov eax,E_OUTOFMEMORY
  ret
 .endif
```

Выделенную память необходимо инициализировать. Указатель на выделенный блок памяти остался в регистре eax; первый DWORD по этому адресу должен быть указателем на виртуальную таблицу реализации методов нашего объекта (Vtbl1):

```
 lea ecx,Vtbl1
 mov [eax],ecx ; указатель на виртуальную таблицу
```

Второй DWORD содержит счетчик ссылок на экземпляр данного объекта; при создании заносим сюда 0:

```
 push 0
 pop [eax+4] ; счетчик нового объекта
```

Третий DWORD - идентификатор объекта, в качетве которого выступает счетчик созданных объектов фабрики классов:

```
 inc objcnt ; еще один объект
 push objcnt
```

```
  pop [eax+8] ; идентификатор нового объекта
```

Осталось вернуть указатель на вновь созданный объект клиенту, который для этой цели передал через аргумент ppvObject адрес соответствующей переменной (в начале метода мы скопировали его в ebx), а также увеличить счетчик ссылок на наш объект на 1:

```
 mov [ebx],eax ; записать указатель на объект
   ; по данному клиентом адресу
 push eax ; указатель 'this'
 call IUnknown_AddRef
 mov eax,S_OK
 ret
IClassFactory_CreateInstance endp
```

Реализация метода LockServer осталась без изменений:

```
IClassFactory_LockServer proc thisptr:DWORD,fLock:DWORD
 .if fLock
  inc counter
 .else
  dec counter
 .endif
 invoke wsprintf,offset buf,offset ms5,counter
 invoke MessageBox,0,offset buf,offset app,0
 mov eax,S_OK
 ret
IClassFactory_LockServer endp
```

Перейдем к реализации IUnknown нашего компонента. Как и в реализации фабрики классов, методы отображают простые окна сообщений, по наличию которых можно судить об их вызове.

```
IUnknown_QueryInterface proc uses ebx esi thisptr:DWORD,
iid:DWORD,ppvObject:DWORD
 mov ebx,thisptr
 mov esi,ppvObject ; адрес переменной клиента, в которую
    ; необходимо возвратить указатель интерфейса
 invoke wsprintf,offset buf,offset ms6,dword ptr [ebx+8] ; id объекта
 invoke MessageBox,0,offset buf,offset app,0
```

Проверка переданных параметров:

```
 .if ppvObject==0
  mov eax,E_INVALIDARG
  ret
 .endif
 invoke IsEqualGUID,offset IUnk,iid
 .if eax==0  ; не IUnknown
  mov [esi],eax
  mov eax,E_NOINTERFACE
  ret
 .endif
```

Сюда попадаем, если затребован интерфейс IUnknown; возвращаем указатель на текущий объект (т.е. "this"), просто увеличив его счетчик ссылок. Указатель на объект был сохранен в регистре ebx, а в регистре esi содержится адрес переменной клиента, в которую надо записать возвращаемое значение:

```
 mov [esi],ebx  ; указатель на текущий объект возвращаем
    ; по адресу, указанному клиентом
 push ebx  ; (указатель 'this')
 call IUnknown_AddRef
 mov eax,S_OK
 ret
IUnknown_QueryInterface endp
```

Перейдем к оставшимся двум методам. Их особенность в том, что теперь счетчик объекта располагается в выделенной памяти, и обращаться к ней необходимо через указатель "this". Кроме того, Release должна уничтожить текущий объект при достижении счетчиком значения 0; это

осуществляется путем освобождения выделенной ранее под структуру объекта памяти через тот же указатель "this".

```
IUnknown_AddRef proc uses ebx thisptr:DWORD
 mov ebx,thisptr
 inc dword ptr [ebx+4]  ; счетчик
 invoke wsprintf,offset buf,offset ms7,dword ptr [ebx+8],dword ptr [ebx+4]
 invoke MessageBox,0,offset buf,offset app,0
 mov eax,dword ptr [ebx+4] ; значение счетчика
 ret
IUnknown_AddRef endp

IUnknown_Release proc uses ebx thisptr:DWORD
 mov ebx,thisptr
 dec dword ptr [ebx+4]  ; счетчик
 invoke wsprintf,offset buf,offset ms8,dword ptr [ebx+8],dword ptr [ebx+4]
 invoke MessageBox,0,offset buf,offset app,0
 .if dword ptr [ebx+4]==0
  invoke LocalFree,ebx ; удалить объект
  dec objcnt
  xor eax,eax
  ret
 .endif
 mov eax,dword ptr [ebx+4] ; значение счетчика
 ret
IUnknown_Release endp

end
```

Def-файл и опции ассемблирования такие же, как ранее. Для экспериментов не забудьте изменить значение пути к серверу в реестре.

Exe-серверы

Внепроцессные серверы COM оформляются в виде exe-модулей. Обычно они представляют собой построенные из компонентов самостоятельные приложения (такие как Word или Excel), доступ к которым можно получить с помощью интерфейсов COM. Такое приложение может быть запущено как обычным способом (например, двойным щелчком на названии соответствующего exe-файла в проводнике), так и системой COM с помощью содержащихся в реестре данных. Чтобы приложение могло различать эти два способа запуска, COM запускает приложение с аргументом командной строки '-Embedding'.

Чтобы убедиться в этом, создадим простейшее приложение (COM_9.asm):

```
.386
.model flat,stdcall
option casemap:none

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.data

app db "CmdLine",0

.code

start:
 invoke GetCommandLine
 mov ecx,eax
 invoke MessageBox,0,ecx,offset app,0
 invoke ExitProcess,0
end start
```

Это приложение отображает в окне сообщения командную строку, с помощью которой оно было запущено. Запишем в подключе 'LocalServer32' нашего "хулиганского" ключа в реестре путь к построенному exe-файлу. Затем запустим наш клиент (COM_2.exe), пометив галочкой "Local server". Как видите, к пути файла добавлен аргумент '-Embedding'. (Наш клиент на время зависнет, как и в случае любого не-COM приложения).

Построим внепроцессный вариант нашего простейшего dll-сервера. Exe-сервер по сравнению с dll-сервером имеет несколько особенностей.

Во-первых, поскольку exe-сервер находится в собственном процессе, он сам должен инициализировать библиотеку COM, вызвав функцию CoInitialize(Ex):

```
invoke CoInitialize,0
.if eax!=S_OK
 invoke MessageBox,0,offset err1,offset app,MB_ICONERROR
 jmp errexit
.endif
```

Во-вторых, он должен создать и зарегистрировать объекты класса для компонентов, которые поддерживает. Для этого используется функция CoRegisterClassObject со следующими аргументами:

• адрес структуры, содержащей CLSID компонента, для которого создается фабрика классов;
• указатель интерфейса созданного объекта фабрики классов;
• тип сервера;
• флаги, указывающие, какой тип соединения с данным сервером поддерживается (например, могут ли сразу несколько клиентов одновременно вызывать объект фабрики классов);
• адрес переменной, в которой будет возвращен регистрационный номер (он используется для последующего освобождения объекта класса).

На этот раз CLSID создаваемого компонента указывается при регистрации сервера явным образом, поэтому халявы, как в прошлый раз, не будет. Придется вшить CLSID компонента в код (используем наш старый {00000000-0000-0000-0000-000000000001}; он размещен в переменной clsid).

```
invoke CoRegisterClassObject,ADDR clsid,ADDR pVtbl,
CLSCTX_LOCAL_SERVER,REGCLS_SINGLEUSE,ADDR rgstr
.if eax!=S_OK
 invoke MessageBox,0,offset err2,offset app,MB_ICONERROR
 jmp errexit
.endif
```

В-третьих, exe-сервер должен иметь цикл обработки сообщений независимо от того, создает он окна или нет. Дело в том, что модель многопоточности COM реализована через механизм т.н. апартаментов: любой объект COM может находиться лишь в одном апартаменте. При инициализации COM создает апартамент для того потока, который вызывает CoInitialize(Ex). Апартамент, в свою очередь, создает скрытое окно, которому посылаются сообщения при вызове методов объекта из других апартаментов (в т.ч. и из других процессов). Именно таким образом (через очередь сообщений потока) осуществляется принудительная синхронизация доступа различных потоков к одному объекту в том случае, если этот объект не создан специально многопоточным. Поэтому любой поток (как клиента, так и сервера), который вызывает CoInitialize(Ex), должен иметь цикл обработки сообщений, содержащий функцию DispatchMessage.

```
mnloop:
 invoke GetMessage,ADDR m,0,0,0
 cmp eax,0
 je exit
 invoke DispatchMessage,ADDR m
 jmp mnloop
exit:
 invoke CoUninitialize
```

```
 invoke MessageBox,0,offset ms,offset app,0
 invoke ExitProcess,m.wParam
errexit:
 invoke ExitProcess,-1
 ret
```

В-четвертых, функция CoRegisterClassObject при регистрации объекта в системной таблице вызывает через интерфейс IClassFactory объекта AddRef. Поэтому необходимо вести отдельный счетчик созданных фабрикой классов объектов и замков сервера, по достижении 0 которым вызывается функция CoRevokeClassObject. Этой функции передается единственный аргумент - регистрационный номер, полученный при вызове CoRegisterClassObject для соответствующего объекта.

Наш exe-сервер реализован "ленивым" способом: фабрика классов не создает новые объекты "компонента" (реализующего, как и в случае с dll-сервером, единственный интерфейс IUnknown), а лишь увеличивает счетчик ссылок единственного статического псевдообъекта, который к тому же имеет общий с замком сервера счетчик. При достижении этим счетчиком значения 0 (при вызове либо метода IUnknown_Release "объекта" компонента, либо метода IClassFactory_LockServer объекта класса) нужно отменить регистрацию фабрики классов. Вот реализации соответствующих методов:

```
IClassFactory_LockServer proc thisptr:DWORD,fLock:DWORD
 .if fLock
  invoke wsprintf,offset buf,offset ms5_1,objcount
  invoke MessageBox,0,offset buf,offset app,0
  inc objcount
 .else
  invoke wsprintf,offset buf,offset ms5_2,objcount
  invoke MessageBox,0,offset buf,offset app,0
  dec objcount
  jnz a2
  ; здесь objcount==0; удалить последнюю ссылку
  ; на IClassFactory, отменив ее регистрацию
  invoke CoRevokeClassObject,rgstr
a2:
 .endif
 mov eax,S_OK
 ret
IClassFactory_LockServer endp

IUnknown_Release proc thisptr:DWORD
 invoke wsprintf,offset buf,offset ms9,objcount
 invoke MessageBox,0,offset buf,offset app,0
 dec objcount
 jnz d1
 ; здесь objcount==0; удалить последнюю ссылку
 ; на IClassFactory, отменив ее регистрацию
 invoke CoRevokeClassObject,rgstr
d1:
 mov eax,objcount
 ret
IUnknown_Release endp
```

В-пятых, при удалении последней ссылки на объект класса для завершения работы сервера необходимо выйти из цикла обработки сообщений. Для этого посылается сообщение WM_QUIT; однако, наш сервер не создавал собственных окон, поэтому сообщение должно быть послано потоку (параметр дескриптора окна функции PostMessage равен 0):

```
IClassFactory_Release proc thisptr:DWORD
 invoke wsprintf,offset buf,offset ms3,counter
 invoke MessageBox,0,offset buf,offset app,0
 dec counter
 jnz e1
 ; поскольку нет явно созданного окна,
 ; послать сообщение о завершении потоку
 invoke PostMessage,0,WM_QUIT,0,0
```

```
e1:
 mov eax,counter
 ret
IClassFactory_Release endp
```

В остальном реализация данного exe-сервера повторяет реализация нашего dll-сервера. Полностью исходные файлы можно найти в каталоге COM_10 архива ComKit2.rar.

Однако, тема нашей статьи не построение exe-серверов, а dll как основа COM, что мы и можем обнаружить в экспериментах с помощью только что созданной утилиты. Построив exe-модуль и записав путь к нему в подключ "LocalServer32", можно запустить нашего клиента (COM_2.exe) и посмотреть, каким образом осуществляется вызов методов внепроцессного сервера. Если до этого вы не имели дела с exe-серверами COM, вы будете удивлены обилием вызовов, которые поступают к нему от системы. Все дело в том, что в дело вступают очередные многочисленные посредники. Попробуем это выяснить.

Для этого реализуем в нашем компоненте простейший нестандартный интерфейс, назвав его, скажем, IFoo. Впрочем, название, как вы уже знаете, не играет особой роли; интерфейс должен иметь уникальный IID. В наших славных традициях присвоим ему IID {00000000-0000-0000-0000-000000000002}. :) Переделка потребуется самая минимальная (COM_11.asm):

• в области данных добавать IID интерфейса:

```
IFoo db 15 dup(0)
 db 2 ; IID {00000000-0000-0000-0000-000000000002}
```

• добавить одно смещение для дополнительного метода в виртуальной таблице для компонента:

```
Vtbl0 DWORD offset IFoo_QueryInterface
 DWORD offset IFoo_AddRef
 DWORD offset IFoo_Release
 DWORD offset IFoo_Member1 ; новый метод
```

• переделать код проверки IID интерфейса в методе QueryInterface для интерфейса компонента:

```
 invoke IsEqualGUID,iid,offset IUnk
 .if eax==0 ; not IUnknown
  invoke IsEqualGUID,iid,offset IFoo
  .if eax==0 ; none of 2 interfaces
   mov [ebx],eax
   invoke MessageBox,0,offset ms7_3,offset app,0
   mov eax,E_NOINTERFACE
   ret
  .else  ; IFoo
   invoke MessageBox,0,offset ms7_2,offset app,0
  .endif
 .else ; IUnknown
  invoke MessageBox,0,offset ms7_1,offset app,0
 .endif
```

• и, наконец, добавить реализацию самого дополнительного метода:

```
IFoo_Member1 proc thisptr:DWORD
 invoke MessageBox,0,offset fooms,offset app,0
 mov eax,S_OK
 ret
IFoo_Member1 endp
```

Строим модуль, записываем путь к нему в подключе LocalServer32 и запускаем клиента. Сначала запросим у компонента интерфейс IUnknown, чтобы убедиться, что все построено правильно и ошибок нет. Если все нормально, запрашиваем IID для нашего IFoo. Ошибка клиента! И по коду (80004002h) мы можем определить, что это E_NOINTERFACE. Но как может отсутствовать интерфейс, который мы реализовали собственными руками (или головой?)?

"Интерфейсная" часть объекта, как уже говорилось, всегда реализуется в dll. И если мы ее не строили, это не значит, что ее нет - она предоставляется системой. То, что мы в своем exe-сервере тоже реализовали "интерфейсную" часть, это явление вторичное - этот интерфейс нужен лишь для взаимодействия с системой COM стандартным образом; внепроцессный сервер может быть реализован вообще без использования интерфейсов COM, по крайней мере, нестандартных. Для этого ему нужно реализовать так называемый нестандартный маршалинг. Но это отдельная тема, и сейчас мы говорить о ней не будем.

Если же внепроцессный сервер не использует нестандартный маршалинг, как в нашем случае, система обеспечивает его средствами стандартного маршалинга. Однако и здесь есть нюанс: система может обеспечить полную автоматизацию маршалинга лишь для тех интерфейсов, которые ей известны, т.е. стандартных. Если интерфейс нестандартный, как наш IFoo, система пытается найти средства для поддержки этого интерфейса, реализованные разработчиком интерфейса или использующего его сервера, через реестр.

Чтобы лучше узнать об этих запросах, слегка модернизируем наш сервер (COM_12.asm). Вероятно, вы обратили внимание, что система запрашивала у сервера множество посторонних интерфейсов. Попробуем узнать, что это за интерфейсы, изменив соответствующий участок IFoo_QueryInterface следующим образом (и заодно убрав лишние окна сообщений, оставив их лишь для QueryInterface):

```
invoke IsEqualGUID,iid,offset IFoo
.if eax==0 ; none of 2 interfaces
 mov [ebx],eax
 invoke StringFromCLSID,iid,offset wbufptr
 invoke WideCharToMultiByte,CP_ACP,0,wbufptr,-1,offset iidbuf,128,0,0
 invoke lstrcpy,offset buf,offset ms7_3
 invoke lstrcat,offset buf,offset iidbuf
 invoke MessageBox,0,offset buf,offset app,0
 mov eax,E_NOINTERFACE
 ret
```

Следующий после IUnknown интерфейс, который система запрашивает у внепроцессного объекта, - IMarshal. Именно этот интерфейс позволяет осуществлять нестандартный маршалинг, и реализуя его, компонент заявляет системе, что он сам будет осуществлять маршалинг своих интерфейсов. Если же система в ответ на свой запрос получает E_NOINTERFACE, она пытается выяснить, есть ли у компонента собственный внутрипроцессный обработчик (запрашивая интерфейс IStdMarshalInfo). Выяснив, что и он не реализован, система окончательно переключается на стандартный маршалинг.

Стандартный маршалинг

Вот тут и вступает в действие раздел реестра "HKEY_CLASSES_ROOT\Interface", который мы обсуждали в прошлой статье. Система ищет в данном разделе ключ, соответствующий IID запрошенного интерфейса, и если не находит, мы получаем ответ E_NOINTERFACE. Если же такой ключ существует, возможны два варианта. Если этот ключ содержит подключ ProxyStubClsid32 (ProxyStubClsid для 16-разрядной версии), его значение по умолчанию является CLSID компонента, который служит для маршалинга данного интерфейса. Если же такого подключа нет, система пытается использовать свой собственный маршалер по умолчанию; если он не поддерживает данный интерфейс, также выдается ошибка E_NOINTERFACE.

Посмотрим, чего хотят от этого компонента. Напишем очередной фиктивный dll-сервер (COM_13.asm):

```
.386
.model flat,stdcall
option casemap:none

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
```

```
include \masm32\include\ole32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\ole32.lib

.data

app  db "ProxyStub32",0
clsms  db "Requested CLSID:",13,10,0
iidms  db "Requested interface:",13,10,0
crlf  db 13,10,0
ms1  db "Dll is loading",0
ms2  db "Dll can unload",0

.data?
wbuf  db 256 dup (?)
iidbuf db 128 dup (?)
buf  db 256 dup (?)
wbufptr DWORD ?

.code

DllGetClassObject proc rclsid:DWORD, riid:DWORD, ppv:DWORD
 invoke StringFromCLSID,rclsid,offset wbufptr
 invoke WideCharToMultiByte,CP_ACP,0,wbufptr,-1,offset iidbuf,128,0,0
 invoke lstrcpy,offset buf,offset clsms
 invoke lstrcat,offset buf,offset iidbuf
 invoke lstrcat,offset buf,offset crlf
 invoke StringFromCLSID,riid,offset wbufptr
 invoke WideCharToMultiByte,CP_ACP,0,wbufptr,-1,offset iidbuf,128,0,0
 invoke lstrcat,offset buf,offset iidms
 invoke lstrcat,offset buf,offset iidbuf
 invoke MessageBox,0,offset buf,offset app,0
 xor ecx,ecx
 lea eax,ppv
 mov [eax],ecx
 mov eax,E_OUTOFMEMORY
 ret
DllGetClassObject endp

DllCanUnloadNow proc
 invoke MessageBox,0,offset ms2,offset app,0
 mov eax,S_OK
 ret
DllCanUnloadNow endp

end
```

Def-файл и опции ассемблирования те же, что использовались ранее при построении dll-сервера. При вызове DllGetClassObject эта функция отображает окно сообщения, указав запрошенные у нее CLSID и IID и каждый раз возвращая ошибку. Для внедрения этой dll в систему придется опять повозиться с реестром.

В разделе "HKEY_CLASSES_ROOT\Interface" необходимо создать ключ с IID нашего интерфейса ({00000000-0000-0000-0000-000000000002}). Под ним создадим еще два подключа: "NumMethods" со значением 4 и "ProxyStubClsid32" со значением CLSID компонента-маршалера нашего интерфейса; в качестве последнего выберем {00000000-0000-0000-0000-000000000003}. Такой же ключ нужно теперь создать и в разделе "HKEY_CLASSES_ROOT\CLSID", указав в подключе "InprocServer32" полный путь к построенной dll.

Вот теперь снова пробуем вызвать интерфейс IFoo. Как видим, после всех запросов интерфейсов у exe-сервера очередь доходит и до нашего ProxyStub - запрашивается интерфейс, по IID которого можно узнать, что это IPSFactoryBuffer.

Интерфейс IPSFactoryBuffer является своего рода фабрикой классов для объектов двух типов: представителей и заглушек нестандартных интерфейсов. Помимо методов IUnknown, этот

интерфейс имеет еще два метода: CreateProxy и CreateStub, создающие соответствующие типы объектов.

Создадим "болванку" для интерфейса IPSFactoryBuffer (COM_14.asm):

```
.386
.model flat,stdcall
option casemap:none

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\ole32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\ole32.lib

.data
; объект класса
pVtbl  DWORD offset Vtbl
counter DWORD 0

; виртуальная таблица для интерфейса IPSFactoryBuffer
Vtbl DWORD offset IPSFactoryBuffer_QueryInterface
 DWORD offset IPSFactoryBuffer_AddRef
 DWORD offset IPSFactoryBuffer_Release
 DWORD offset IPSFactoryBuffer_CreateProxy
 DWORD offset IPSFactoryBuffer_CreateStub

; IID интерфейсов
IUnk  dd 0   ; IID для IUnknown
  dw 0
  dw 0
  db 0C0h,0,0,0,0,0,0,46h
IPSFB  dd 0D5F569D0h ; IID для IPSFactoryBuffer
  dw 593Bh
  dw 101Ah
  db 0B5h,69h,8,0,2Bh,2Dh,0BFh,7Ah

app  db "ProxyStub32",0
ms2  db "Dll can unload",0
msproxy db "CreateProxy requested",0
msstub db "CreateStub requested",0
msiunk db "QueryInterface:",13,10,"IUnknown requested",0
msipsfb  db "QueryInterface:",13,10,"IPSFactoryBuffer requested",0
msnone db "QueryInterface:",13,10,"other interface requested",0

.code

DllGetClassObject proc rclsid:DWORD, riid:DWORD, ppv:DWORD
 ; Проверка интерфейса осуществляется в функции
 ; QueryInterface, которой этот вызов и перенаправляется
 push ppv
 push riid
 lea eax,pVtbl
 push eax ; this
 call IPSFactoryBuffer_QueryInterface
 ret ; переправить значение, возвращенное QueryInterface
DllGetClassObject endp

DllCanUnloadNow proc
 .if counter>0
  mov eax,S_FALSE
 .else
  invoke MessageBox,0,offset ms2,offset app,0
  mov eax,S_OK
 .endif
 ret
DllCanUnloadNow endp

IPSFactoryBuffer_QueryInterface proc uses ebx,thisptr:DWORD,
```

```
iid:DWORD,ppvObject:DWORD
 .if ppvObject==0 ; неверный параметр
  mov eax,E_INVALIDARG
 .endif
 mov ebx,ppvObject
 ; принимает IID лишь для IUnknown и IPSFactoryBuffer
 invoke IsEqualGUID,iid,offset IUnk
 .if eax==0  ; не IUnknown
  invoke IsEqualGUID,iid,offset IPSFB
  .if eax==0 ; ни один из 2 интерфейсов
   mov [ebx],eax
   invoke MessageBox,0,offset msnone,offset app,0
   mov eax,E_NOINTERFACE
   ret
  .else  ; IPSFactoryBuffer
   invoke MessageBox,0,offset msipsfb,offset app,0
  .endif
 .else   ; IUnknown
  invoke MessageBox,0,offset msiunk,offset app,0
 .endif
 lea eax,pVtbl
 mov [ebx],eax
 push eax ; 'this'
 call IPSFactoryBuffer_AddRef
 mov eax,S_OK
 ret
IPSFactoryBuffer_QueryInterface endp

IPSFactoryBuffer_AddRef proc thisptr:DWORD
 inc counter
 mov eax,counter
 ret
IPSFactoryBuffer_AddRef endp

IPSFactoryBuffer_Release proc thisptr:DWORD
 dec counter
 mov eax,counter
 ret
IPSFactoryBuffer_Release endp

IPSFactoryBuffer_CreateProxy proc thisptr:DWORD,pUnkOuter:DWORD,
iid:DWORD,ppProxy:DWORD,ppv:DWORD
 invoke MessageBox,0,offset msproxy,offset app,0
 mov eax,E_NOINTERFACE
 ret
IPSFactoryBuffer_CreateProxy endp

IPSFactoryBuffer_CreateStub proc thisptr:DWORD,iid:DWORD,
pUnkServer:DWORD,ppStub:DWORD
 invoke MessageBox,0,offset msstub,offset app,0
 mov eax,E_NOINTERFACE
 ret
IPSFactoryBuffer_CreateStub endp

end
```

Чтобы этот сервер запускался, необходимо изменить путь в соответствующем подключе "InprocServer32".

Как можно видеть, ProxyStub - обычный внутрипроцессный сервер COM, реализующий другой тип фабрики классов. Чтобы и дальше проследить действия системы COM, можно было бы реализовать методы CreateProxy и CreateStub по-настоящему, но здесь мы уже выходим из царства COM и вступаем в царство RPC, а это совсем другой разговор.

COM повсюду. Или как использовать регулярные выражения при программировании на ассемблере

Автор: W[4Fh]LF

Дата: 2007-02-17

1. Для чего мы здесь сегодня собрались или о чём на самом деле эта статья?

Навеяло недавней статьёй о COM с этого сайта и его использовании в ассемблере. Всплыла у меня в голове старая задумка: разобрать интерфейс VBScript Regular Expressions 5.5 и использовать все его потрясающие возможности при программировании на ассемблере.

На самом деле, назвать потрясающими возможности этого препроцессора можно с трудом, т.к. он не является самым мощным из всех препроцессоров, скорее являются потрясающими возможности самих регулярных выражений, как средства работы со строковыми данными, которое рано или поздно должно было быть изобретено. Познакомившись с ними уже довольно давно, я по-настоящему стал фанатом регулярных выражений и на данный момент с трудом представляю себе, как можно работать с больших объёмом текстовых данных стандартными функциями, или как можно работать в большом объёме, с любыми объёмами текстовых данных, без использования регулярок. Тем не менее, как показывает практика, очень многие люди до сих пор не пользуются возможностями регулярных выражений в тех случаях, когда их использование даёт огромные плюсы и преимущества. Прежде всего нетривиальные задачи, которые описываются и решаются с помощью кучи циклов, функций и прочих конструкций языка, могут быть описаны и решены очень просто с помощью регулярного выражения(далее просто выражения). Понятно, что это позволяет экономить кучу времени, объёма кода и, что очень важно, нервов программиста. Вдобавок ко-всему прочему, не стоит забывать, что речь идёт о программировании на ассемблере, где все вышеперечисленные плюсы заметны гораздо сильнее. Нет, я конечно понимаю, что мы любим ассемблер за отсутствие каких-либо удобств и предоставления права выбора и контроля всех ситуаций нам. Однако, я считаю, что данный материал будет полезен, в конце-концов, если вы являетесь ярым фанатиком низкого уровня, можете закрыть эту страницу или всё-таки прочесть и иметь право выбора, а главное представление о том, что это такое - регулярные выражения(если такого представления на данный момент вы не имеете).

2. О предмете наших рассуждений или ради чего мы всё это делаем?

Тот, кто знаком с понятием регулярное выражение может смело пропустить эту часть статьи. Разбор синтаксиса регулярных выражений и демонстрирование всех его возможностей - это тема отдельной статьи, если не книги. Ссылки на литературу я дам ниже, здесь же я хотел немного рассказать о том, что это такое и какие проблемы призваны решать выражения. В совокупности, написание регулярных выражений можно назвать программированием на отдельном, небольшом языке программирования, которое в общем случае призвано решать одну задачу: проверка соответствия какой-либо части или всего текста регулярному выражению. Можно сказать, вы задаёте универсальную маску для поиска соответствий в тексте, а на выходе имеете результат: True или False. Конечно, со временем эта возможность обросла другими полезными методами, которые мы рассмотрим в этой статье. Самый простой пример. Допустим, есть у нас некоторая строка, мы хотим проверить, является ли эта строка шестнадцатеричным числом, введённым в С-подобном формате. Ну, представили, сколько это строк на ассемблере?:)

А теперь выражение: ^0x[0-9A-Fa-f]{1,8}&

Я знаю, что поначалу для людей незнакомых с регулярными выражениями эта строка может показаться загадкой. Эта маленькая строчка полностью исключает возможность ввода несоответствующего формату числа. Давайте разберём его, дабы вы поняли, что не всё так

страшно: ^ - Этот метасимвол означает начало строки. 0x - эта последовательность говорит о том, что сразу после начала строки должна следовать эта пара символов. Проще говоря, наша строка должна начинаться с этих двух символов. [0-9A-Fa-f] - эта конструкция называется символьным классом. Она определяет, какой символ может идти сразу за парой 0x, а говорит она о том, что после этой пары может идти любой символ из диапазона 0-9 = 0123456789, A-F = ABCDEF, a-f = abcdef. {1,8} - говорит о том, что символ из диапазона, который определён символьным классом ранее, должен повториться от 1 до 8 раз. & - конец строки.

А теперь прочтём это выражение: Вначале строки должны идти символы 0x, сразу за ними должны следовать 1-8 символов из числа тех, которые характеризуют цифры шестнадцатеричной системы, после чего строка должна заканчиваться и не содержать больше в себе ничего. При этом хочу сделать оговорку, если выражение не содержит в себе метасимволов начала и конца строки(^$), то ищется соответствующее маске значение по всём тексте, т.е. текст стоящий до и после числа, значения иметь не будет.

Давайте представим другую ситуацию: допустим, мы знаем, что человек может написать:

• Сегодня 0 градусов Цельсия/Фаренгейта
• Сегодня ноль градусов Цельсия/Фаренгейта
• Сегодня нуль градусов Цельсия/Фаренгейта

Итого мы имеем 6 вариантов правильной последовательности, что бы вы сделали? Вызвали 6 раз lstrcmp? А если бы я сказал, что регистр не играет роли? Ок, ещё пару вызовов функции для преобразования к одному регистру. А если бы я сказал, что между словами могут быть несколько пробелов? Вы бы сказали, что тогда лучше будет использовать поиск по словам, но и это бы выглядело громоздко и довольно некрасиво.

Теперь внимание, регулярное выражение, которое описывает все эти варианты:

"Сегодня +(0|ноль|нуль) +градусов +(Цельсия|Фаренгейта)"

Впечатляет?:) Должно впечатлять.;)

Вы скажете: "Где здесь учитывается непостоянство регистра?", - Мы могли бы конечно это учесть в самом выражении, оно бы выросло ровно в два раза и выглядело не так элегантно. К счастью, для этого препроцессоры обычно предусматривают специальные флаги, о которых будет сказано ниже.

Теперь самое время сказать и о недостатках регулярных выражений. А, собственно, какие недостатки? Недостаток только один: такая гибкость и универсальность налагает определённые условия на время обработки. Нет, за все движки не скажу, но что касается VBScript Regular Expressions, то с уверенностью могу сказать, что он очень хорошо оптимизирован. Само собой реализация под конкретный формат, с небольшим числом вариаций, будет работать быстрее, да собственно оно и понятно, так и должно быть. В общем же случае, я думаю все уже поняли, что возможности, которые предоставляются, позволяют забыть о скорости и в большинстве случаев регулярные выражения всё-таки должны преобладать в задачах направленных на обработку текста. Тем более, что, как правило, задачи которые призваны решать проблемы обработки текста по сложным правилам, в итоге реализуются громоздкими функциями, которые не всегда не то, что быстрее регулярных выражений, а наоборот.

3. Собственно библиотека и разбор интерфейса

Сам компонент VBScript.RegExp находится в библиотеке vbscript.dll, которая поставляется с IE(начиная с 4 версии и выше). Описание методов и свойств этого объекта можно прочесть в MSDN, просмотреть все методы и их id можно с помощью дополнительных утилит(например "OLE/COM Object Viewer" из комплекта VS), но никто нам не говорит о смещениях этих методов в так называемой виртуальной таблице методов объекта, именуемой vftable, о ней будет сказано позже. Языки высокого уровня позволяют абстрагироваться от этого, но чтобы нам получить

желаемый результат придётся всё-таки лезть в дебри и исследовать методы вызова. Походу дела я, конечно, буду объяснять, что мы делаем и в чём суть, но всё-таки предполагается, что вы уже знакомы с COM, если нет, то настоятельно рекомендую заглянуть в раздел литература.

Сначала нужно создать копию объекта в памяти, запросить его интерфейс, получить указатель на управляющую таблицу. Создаваемый нами класс именуется VBScript.RegExp и имеет свой идентификатор(CLSID): {3F4DACA4-160D-11D2-A8E9-00104B365C9F} Подробнее описание этого класса можно посмотреть в разделе реестра: HKEY_CLASSES_ROOT\CLSID\{3F4DACA4-160D-11D2-A8E9-00104B365C9F}

На основе всего этого мы имеем достаточно данных, чтобы создать объект и получить его интерфейс, в этом нам поможет следующая функция:

```
.data
;VBScript.RegExp
GUID_RegExp db 0A4h, 0ACh, 04Dh, 03Fh, 00Dh, 016h, 0D2h, 011h, \
     0A8h, 0E9h, 000h, 010h, 04Bh, 036h, 05Ch, 09Fh
;IUnknown
GUID_I   db 000h, 000h, 000h, 000h, 000h, 000h, 000h, 000h, \
        0C0h, 000h, 000h, 000h, 000h, 000h, 000h, 046h

IUnknown   _MULTI_QI

.code

;Set RegExp = New RegExp(GUID_RegExp, *_MULTI_QI)
CreateInterface proc
 invoke CoInitialize,0
 push offset GUID_I
 pop IUnknown.pIID
 invoke CoCreateInstanceEx, offset GUID_RegExp, 0, 5, 0, 1, \
offset IUnknown
 ret
CreateInterface endp
```

invoke CoInitialize,0 - инициализирует библиотеку COM в текущем потоке. Следует вызывать всегда, перед работой с COM компонентом.

Ключевым моментом является вызов CoCreateInstanceEx, функции, которая принимает следующие параметры:

• rclsid - CLSID класса, объект которого мы хотим получить.
• punkOuter - интерфейс агрегирующего объекта, в нашем случае агрегирование не применяется, поэтому 0.
• dwClsCtx - контекст, в котором будет создан объект. В нашем случае это сервер внутрипроцессорный локальный, поэтому 5(CLSCTX_INPROC_SERVER+ CLSCTX_LOCAL_SERVER).
• pServerInfo - Информация о компьютере на котором создаётся объект. Т.к. в нашем случае создание объекта идёт локально, то 0.
• cmq - Кол-во структур, в массиве, описывающих создаваемые объекты. Мы хотим создать только один объект, поэтому и передаваемое значение будет равно 1.
• pResults - Массив структур MULTI_QI, каждая из которых определена следующим образом:

```
_MULTI_QI struct
 pIID dd ?  ; Идентификатор интерфейса, который мы хотим получить.
 pItf  dd ?  ; Указатель на интерфейс полученный при запросе. На входе должен
    ; быть равен 0. Точнее это не указатель на интерфейс, это скорее
    ; указатель на некоторую управляющую структуру, о ней чуть ниже.
 hr  dd ? ; Для каждого создаваемого объекта функция CoCreateInstanceEx
   ; запрашивает реализуемый интерфейс, идентификатор которого мы
   ; передали первым параметром в эту структуру. Данное поле есть
   ; результат запроса реализуемого интерфейса. На входе должен быть
   ; равен 0.
_MULTI_QI ends
```

Теперь пару слов о pIID и GUID_I. Дело в том, что какой бы ни был интерфейс у создаваемого объекта, он наследуется от базового интерфейса в COM - IUnknown(его GUID и содержит переменная GUID_I) и должен реализовывать все его методы. А методов всего 3:

Последние два метода предназначены для управления счётчиком ссылок:

• AddRef - функция инкрементирует счётчик ссылок на объект и возвращает его новое значение.
• Release - функция уменьшает счётчик ссылок на объект. При этом, если значение счётчика обратилось в 0, то объект освобождается из памяти.

Первый метод:

QueryInterface - Данная функция запрашивает интерфейс. Передавая в неё первым параметром GIUD IUnknown, в случае, если такой интерфейс реализуется объектом, во втором переданном параметре(это должен быть указатель на DWORD) на выходе мы получаем указатель на запрашиваемый интерфейс. При этом функция вызывает AddRef , тем самым, инкрементируя счётчик ссылок на объект.

Реализацию этих методов вы можете посмотреть в приложении к статье.

Получилось немного сумбурно, поэтому давайте подытожим:

• Нужный нам объект имеет свой идентификатор класса - CLSID.
• Чтобы создать копию объекта в памяти мы вызываем CoCreateInstanceEx, эта функция создаёт копию объекта с указанным CLSID. Так же функция запрашивает реализуемый всеми COM-объектами интерфейс IUnknown, который имеет стандартный GUID. Метод QueryInterface этого интерфейса запрашивает все прочие интерфейсы реализуемые объектом.
• После вызова этого метода(QueryInterface)1 мы имеем в памяти интерфейс IRegExp, имеем мы его в виде некоторой виртуальной таблицы адресов методов, именуемой vftable. Все реализуемые методы вызываются косвенно, т.е. относительно начала этой таблицы. Теперь наша цель узнать смещения методов относительно начала этой таблицы.

1 Этот метод вызывается CoCreateInstanceEx по умолчанию.

4. Методы, их назначение и использование.

Итак, IRegExp имеет несколько свойств, я бы их назвал флагами, которые характеризуют способ восприятия и обработки входных данных:

• Global - Этот флаг говорит препроцессору, что он должен обрабатывать текст целиком, в ином случае текст будет обработан до первого совпадения.
• IgnoreCase - Регистронезависимый режим, тут всё понятно.
• MultiLine - Говорит о том, что входной текст многострочный, если данный флаг установлен не будет, то переносы строк не учитываются.

Каждое из этих свойств устанавливается аналогично другим, с одним лишь отличием: смещение в vftable у каждого метода будет разным. Вычислив смещение всех методов, я определил их как константы:

```
.const
 FLAG_GLOBAL  equ  30h ; RegExp.Global
 FLAG_IGNORECASE equ 28h ; RegExp.IgnoreCase
 FLAG_MULTILINE equ 38h ; RegExp.MultiLine
 CLRFLAG   equ  0
 SETFLAG   equ  -1
```

Последние две константы тоже вопросов вызывать не должны, они устанавливают в False или True соответствующие свойства объекта.

Теперь процедура, которая работает с этими свойствами:

```
;RegExp.Method = Bool
RegExp_Method proc Method:DWORD,Bool:DWORD
 push Bool
 mov eax,IUnknown.pItf
 push eax
 mov ecx,[eax]
 mov edx,Method
 call dword ptr[ecx+edx]
 ret
RegExp_Method endp
```

Процедура принимает два параметра: Method - характеризуется одной из вышеперечисленных констант и является смещением относительно начала vftable; Bool - переменная, которая говорит о нашем намерении установить значение свойства в True или False. Далее, как видно, мы кладём в стек два параметра:

```
push Bool
```

Значение флага.

```
mov eax,IUnknown.pItf
push eax
```

Указатель на некую управляющую структуру, о ней я могу сказать лишь то, что первым параметром этой структуры является указатель на vftable, а четвёртым счётчик ссылок на объект. Вообще, вызов любого метода принимает минимум один параметр - указатель на эту структуру, поэтому данная конструкция будет использоваться везде.

```
mov ecx,[eax]
```

Теперь в ecx указатель на vftable.

```
call dword ptr[ecx+edx]
```

Суммируя смещение vftable со смещением соответствующего метода относительно начала vftable, мы вызываем нужный нам метод.

Хорошо, нужные свойства объекта перед работой мы установили, теперь следует поговорить о методе Pattern, который устанавливает маску поиска. Вызов этого метода реализован следующим образом:

```
;RegExp.Pattern = Expression
RegExp_Pattern proc Expression:DWORD, lpWideCharStr:DWORD, \
cchWideChar:DWORD
 local pBSTR:DWORD

 invoke lstrlen, Expression
 invoke MultiByteToWideChar, CP_ACP, 0, Expression, eax, \
lpWideCharStr, cchWideChar
 invoke SysAllocStringLen, lpWideCharStr, eax
 mov pBSTR,eax
 push eax
 mov eax,IUnknown.pItf
 push eax
 mov ecx,[eax]
 call dword ptr[ecx+20h]
 invoke SysFreeString, pBSTR
 ret
RegExp_Pattern endp
```

• Expression - указатель на ANSI строку, которая содержит маску поиска, т.е. регулярное выражение.
• lpWideCharStr - Указатель на буфер, который будет содержать преобразованную в UNICODE строку Expression.
• cchWideChar - размер буфера в байтах.

Теперь я объясню для чего нам преобразовывать строку в UNICODE. Всё дело в том, что методы IRegExp работают со строками в формате BSTR, который характеризуется UNICODE строкой и двойным словом перед ней, содержащим длину в символах. Этот тип является родным для VB и поэтому там особых проблем с этим нет, однако, чтобы привести к этому типу ANSI строку нужно преобразовать её сначала в UNICODE, а потом вызвать специальную функцию SysAllocStringLen, которая принимает в качестве параметров указатель на UNICODE строку и её длину в символах, а возвращает указатель на BSTR в памяти.

```
 invoke lstrlen, Expression
 invoke MultiByteToWideChar, CP_ACP, 0, Expression, eax, \
lpWideCharStr, cchWideChar
 invoke SysAllocStringLen, lpWideCharStr, eax

Преобразование ANSI строки в BSTR.

push eax
 mov eax,IUnknown.pItf
 push eax
```

Вторым параметром у нас идёт указатель на BSTR, который вернула функция SysAllocStringLen, а первый параметр указатель на уже знакомую управляющую структуру объекта.

```
call dword ptr[ecx+20h]
 invoke SysFreeString, pBSTR
```

Опытным путём было установлено, что смещение метода Pattern в vftable равно 20h, вызываем его и освобождаем память от выделенной ранее строки.

На данном этапе мы можем менять свойства, которые влияют на характер обработки текста и можем устанавливать маску поиска. Самое время разобрать методы, которые направлены непосредственно на обработку самого текста. Первым из них является метод Test. Test позволяет сделать поиск совпадений в тексте. Объединив всё нужное в одну функцию получаем:

```
;RegExp.Test(lpData)
RegExp_Test proc lpData:DWORD, lpWideCharStr:DWORD, \
cchWideChar:DWORD
 local Result:WORD
 local pBSTR:DWORD

 invoke lstrlen, lpData
 invoke MultiByteToWideChar, CP_ACP, 0, lpData, eax, lpWideCharStr, \
 cchWideChar
 invoke SysAllocStringLen, lpWideCharStr, eax
 mov pBSTR, eax
 lea ecx,Result
 push ecx
 push eax
 mov eax,IUnknown.pItf
 push eax
 mov ecx,[eax]
 call dword ptr[ecx+40h]

 invoke SysFreeString, pBSTR
 xor eax,eax
 cmp word ptr[Result],0
 je @f
  mov eax,1
 @@:
 ret
RegExp_Test endp
```

• lpData - указатель на данные(ANSI) в которых будет осуществлён поиск.
• lpWideCharStr - указатель на буфер, в который будет помещёна преобразованная в UNICODE строка lpData.
• cchWideChar - размер буфера в байтах.

Действия, которые мы проделываем вначале над буфером lpData Вам уже известный, поэтому их пояснять я не буду, а вот с параметрами, передаваемыми в метод, я немного поясню:

```
lea ecx,Result
 push ecx
 push eax
 mov eax,IUnknown.pItf
 push eax
```

Первым в стек попадает указатель на слово. В это слово, после вызова метода Test, будет возвращён результат проверки текста на соответствие маске. Не знаю почему MS сделали этот параметр равным слову, но это так. Ко всему прочему привычные результаты 0 - False; 1- True тут тоже не имеют силы, в данном случае при успешном поиск в Result будет возвращён -1, а при отсутствии совпадений - 0.

Вторым параметром мы кладём указатель на BSTR, его опять же нам вернула уже знакомая функция SysAllocStringLen. И, наконец, последним в стек уходит указатель на управляющую структуру объекта.

```
 call dword ptr[ecx+40h]

 invoke SysFreeString, pBSTR
 xor eax,eax
 cmp word ptr[Result],0
 je @f
  mov eax,1
 @@:
 ret
```

Вызываем метод, его смещение относительно начала таблицы равно 40h, далее освобождаем память и обрабатываем результат. Чтобы не изменять традициям и не путаться, я сделал всё привычным образом, т.е. в случае успешного поиска совпадений функция в eax вернёт 1, в случае неудачи 0.

В продолжении хочется рассмотреть обёртку над основным методов Test, метод который именуется Replace. Метод ищет совпадения в тексте по маске и заменяет их на другой текст. Иногда бывает очень полезен, поэтому я решил его разобрать. Метод кажется немного нагромождённым и большим, но на самом деле всё просто.

```
;RegExp.Replace(SourceString, ReplaceVar)
RegExp_Replace proc lpData:DWORD, lpWideCharStr:DWORD, \
cchWideChar:DWORD, \
     ReplaceVar:DWORD,  \
ResultLPSTR:DWORD, \
szResultLPSTR:DWORD

 local ReplaceStr:DWORD
 local bReplaceStr:DWORD
 local szReplaceVar:DWORD
 local szWReplaceVar:DWORD

 ;#######Convert ReplaceVar to BSTR string format#######
 invoke lstrlen,ReplaceVar
 mov szReplaceVar,eax
 imul eax,2
 inc eax
 mov szWReplaceVar,eax
 invoke VirtualAlloc, 0, szWReplaceVar, MEM_COMMIT, \
      PAGE_READWRITE
 mov ReplaceStr,eax
 invoke MultiByteToWideChar, CP_ACP, 0, ReplaceVar, szReplaceVar, \
ReplaceStr, szWReplaceVar
 invoke SysAllocStringLen, ReplaceStr, eax
 mov bReplaceStr,eax
 invoke VirtualFree, ReplaceStr, szWReplaceVar, MEM_DECOMMIT
 ;#####################################################
```

```
 invoke lstrlen, lpData
 invoke MultiByteToWideChar, CP_ACP, 0, lpData, eax, lpWideCharStr, \
 cchWideChar
 invoke SysAllocStringLen, lpWideCharStr, eax

 push ResultLPSTR
 push 0
 push bReplaceStr
 push 0
 push 8
 push eax
 mov eax,IUnknown.pItf
 push eax
 mov ecx,[eax]
 call dword ptr[ecx+44h]

 invoke SysFreeString, bReplaceStr
 mov eax,[ResultLPSTR]
 mov eax,[eax]
 invoke WideCharToMultiByte, CP_ACP, 0, eax, -1, ResultLPSTR, \
szResultLPSTR, 0, 0
 ret
RegExp_Replace endp
```

• lpData - Указатель на данные в которых будет осуществлён поиск совпадений и при необходимости производиться замена.
• lpWideCharStr - Буфер для преобразования lpData в UNICODE.
• cchWideChar - размер буфера lpWideCharStr в байтах.
• ReplaceVar - указатель на ANSI строку которой будут заменены все совпадения в тексте.
• ResultLPSTR - указатель на результирующий буфер. Туда будет записан результат в формате ANSI
• szResultLPSTR - размер буфера в байтах.

Теперь давайте со всем этим добром по-порядку разбираться:

```
invoke lstrlen,ReplaceVar
 mov szReplaceVar,eax
 imul eax,2
 inc eax
 mov szWReplaceVar,eax
 invoke VirtualAlloc, 0, szWReplaceVar, MEM_COMMIT, \
      PAGE_READWRITE
 mov ReplaceStr,eax
```

Прежде всего нужно выделить буфер и привести RaplaceVar к типу BSTR. Эта часть кода вычилсяет требуемый размер буфера и выделяет память под него.

```
invoke MultiByteToWideChar, CP_ACP, 0, ReplaceVar, szReplaceVar, \
ReplaceStr, szWReplaceVar
 invoke SysAllocStringLen, ReplaceStr, eax
 mov bReplaceStr,eax
 invoke VirtualFree, ReplaceStr, szWReplaceVar, MEM_DECOMMIT
```

После чего мы преобразовываем ANSI строку на которую указывает ReplaceVar в формат UNICODE, далее приводим её к типу BSTR и освобождаем выделенную ранее память, т.к. буфер с UNICODE представлением ReplaceVar нам больше ненужен. В результате bReplaceStr указывает на участок памяти содержащий строку для замены приведённую к типу BSTR.

```
 invoke lstrlen, lpData
 invoke MultiByteToWideChar, CP_ACP, 0, lpData, eax, lpWideCharStr, \
 cchWideChar
 invoke SysAllocStringLen, lpWideCharStr, eax
```

После чего мы проделываем уже знакомые операции над буфером lpData.

```
 push ResultLPSTR  ; 1
 push 0    ; 2
 push bReplaceStr  ; 3
```

```
 push 0   ; 4
 push 8   ; 5
 push eax   ; 6
 mov eax,IUnknown.pItf
 push eax   ; 7
```

Чтобы было удобней, я буду нумеровать параметры в том порядке, в котором они попадают в стек.

Первым в стек попадает указатель на ResultLPSTR, но после вызова метода Replace он не будет указывать на обработаную строку как может показаться, на самом деле по адресу ResultLPSTR будет лежать указатель на BSTR обработанной строки, я кладу туда его для удобства, т.к. этот указатель всего-лишь промежуточный параметр выделять под него отдельную переменную я не хотел.

Назначение второго, ровно как и четвёртого, параметра мною так и не было установлено, они всегда были равны 0, поэтому я решил не лезть в дебри и смириться.

Третьим в стек попадает указатель на BSTR строки для замены.

Пятый параметр является вероятно каким-то набором флагов, однако и его принадлежность установить не удалось, я всегда передаю туда 8 и пока никаких проблем не возникло, оставим так.

Шестым параметром мы кладём в стек указатель на текст который подлежит обработке, точнее на его "близнеца" приведённого к типу BSTR. Ну и седьмым идёт указатель на управляющую структуру.

```
mov ecx,[eax]
 call dword ptr[ecx+44h]

 invoke SysFreeString, bReplaceStr
```

Далее следует вызов метода и освобождение уже неиспользуемого буфера.

```
mov eax,[ResultLPSTR]
 mov eax,[eax]
 invoke WideCharToMultiByte, CP_ACP, 0, eax, -1, ResultLPSTR, \
szResultLPSTR, 0, 0
```

Ну и заканчивается это всё извелечением указателя на обработанный текст, конвертированием его в ANSI строку, которая будет находится по адресу ResultLPSTR.

5. Механизм в действии или примеры использования

Наверное уже достал всех с описанием всяких методов и интерфейсов, а в деле так ничего и не показал, что ж, сейчас покажу, обо всём по-порядку.

Объединив всё вышесказаное в один файл, именуемый regexp.inc(вы его можете утянуть в качестве приложения к статье) и положив его спокойно в include использовальние выражений становится предельно простым и удобным. Далеко ходить не буду, разберу всё на том же формате представления шестнадцатиричных чисел в C-подобном формате.

```
.586
.model flat,stdcall
option casemap:none

    include windows.inc
    include user32.inc
    include kernel32.inc
    include masm32.inc
    include regexp.inc

    includelib user32.lib
    includelib kernel32.lib

.data?
 ResBuffer  db 512 dup(?)
.data
```

```
 Expressions  db "^0x[A-F0-9a-f]{1,8}$",0
 TrueData  db "0x32dfcfdf",0
 FalseData  db "32dfcfdf",0
 Try    db "True",0
 Fal    db "False",0
.code
start:
 ; Создаём копию объекта и запрашиваем требуемые интерфейсы
 invoke CreateInterface

; Устанавливаем нужные флаги.
 invoke RegExp_Method, FLAG_MULTILINE, SETFLAG
 invoke RegExp_Method, FLAG_GLOBAL, SETFLAG
 invoke RegExp_Method, FLAG_IGNORECASE, SETFLAG

 ; Устанавливаем маску поиска
 invoke RegExp_Pattern, offset Expressions, offset ResBuffer, 512

 ; Осуществляем поиск совпадений маске.
 invoke RegExp_Test, offset TrueData, offset ResBuffer, 512
 test eax,eax
 je @f
  invoke MessageBox, 0, offset Try, 0, 0
 @@:
 invoke RegExp_Test, offset FalseData, offset ResBuffer, 512
 test eax,eax
 jne @f
  invoke MessageBox, 0, offset Fal, 0, 0
 @@:
 ; Уничтожаем копию объекта в памяти и свобождаем библоитеки.
 invoke ReleaseInterface
 invoke ExitProcess, 0
end start
```

А данном примере я вызвал функцию RegExp_Test два раза с одной и той же маской, передавая первый раз валидные данные, а второй раз данные, которые не соответствуют маске. Можете скомпилировать и проверить результат, в обоих случаях на экране появится сообщение с результатом.

Теперь продемонстрирую работу функции RegExp_Replace:

```
.data?
 ResBuffer  db 512 dup(?)
 DataBuffer  db 512 dup(?)
.data
 Expressions  db "0x[A-F0-9a-f]{1,8}",0
 Data   db "одна 0x32dfcfdf, вторая 0xFFFF, а вот это"
    db "неправильное значение 123ABCDEF",0
 rVar   db "цЫферка",0
.code
start:
 invoke CreateInterface

 invoke RegExp_Method, FLAG_MULTILINE, SETFLAG
 invoke RegExp_Method, FLAG_GLOBAL, SETFLAG
 invoke RegExp_Method, FLAG_IGNORECASE, SETFLAG
 invoke RegExp_Pattern, offset Expressions, offset ResBuffer, 512

 invoke RegExp_Replace, offset Data, offset DataBuffer, 512, \
       offset rVar, offset ResBuffer, 512
 invoke MessageBox, 0, offset ResBuffer, 0, 0

 invoke RegExp_Method, FLAG_GLOBAL, CLRFLAG
 invoke RegExp_Replace, offset Data, offset DataBuffer, 512, \
       offset rVar, offset ResBuffer, 512
 invoke MessageBox, 0, offset ResBuffer, 0, 0

 invoke ReleaseInterface
 invoke ExitProcess, 0
end start
```

Прежде всего обратите внимание на то, что я немного изменил само выражение, убрав метасимволы начала и конца строки(^$) я говорю препроцессору, что для меня не имеет значение, что стоит до и после выражения соответствующего маске. На момент вызова первый раз функции RegExp_Replace свойство Global установлено в True, что говорит препроцессору о том, что следует осуществить поиск по всему тексту, т.о. как результат мы видим такое сообщение:

После этого я присваиваю этому свойству значение False и в результате мы видим такое сообщение:

Т.е. текст был обработан до первого соответствия маске, всё, что после просто проигнорировалось.

6. Заключение и благодарности

Ну что ж, не знаю насколько был полезен вышеизложенный материал, однако я надеюсь, что он был интересным. Это мой первый опыт в написании подобных статей и поэтому я прошу отнестись с пониманием.

Хотелось бы выразить благодарность человеку с ником December, это очень отзывчивый и приятный в общении человек, спасибо ему за замечания по статье и за указания нужного направления. Остальные же люди просто залажали статью на стадии, когда она ещё только зарождалась, с чем это связано я так и не понял, однако это не отбило охоту дописать данный материал, наверное прежде всего для себя и благодаря December. Если я и не до конца следовал его наставлениям и исправил не всё, что он заметил, то только потому, что не знал как, в силу своего малого опыта в роли автора.

Литература:

1. Фридл Дж. Регулярные выражения (2-е изд.), Питер 2003.
2. 3 кита COM. Кит первый: реестр.
3. Много полезного материала по практической реализации COM можно подчерпнуть из drkb.ru.
4. Known IUnknown.
5. Получение информации о COM-интерфейсах.

В приложении (находится в архиве `wasm_files.zip`: `files/0449/comregexp.zip`) к статье вы найдёте файл regexp.inc и небольшой пример использования из статьи.

Взгляд на ООП из низкого уровня

Автор: vid, пер. DarkWanderer

Дата: 2008-08-29

Как правило, большинству низкоуровневых программистов объектно ориентированное программирование кажется сложным. В этой статье объясняются некоторые моменты ООП и приводятся примеры их устройства с помощью языков C и Assembler.
Замечу, что примеры не всегда сто процентно читаемы, и не всегда правильны. Примеры не подходят для немедленного выполнения, они служат только иллюстрациями к тексту.
Главное достоинство ООП – возможность получить доступ к объектам посредством сходного интерфейса.
Эта мысль поясняется в остальной части статьи.

Простой объект (First object)

Создадим простой объект. Каким он будет? Хороший пример, когда ООП действительно нужно, это разработка игр. Представьте себе, что мы разрабатываем компьютерную игру. В нашей игре будут враги, оружие, ракеты и т.п. Вещи, которые вы привыкли видеть в компьютерных играх. И все они будут представлены как объекты, которые, в свою очередь, как структуры данных в памяти.
Естественно, типов объектов будет множество. Например, аптечка (medikit), для восстановления здоровья игрока, или ружье, из которого можно стрелять. Объектов какого-либо типа в игре тоже может быть много. Все эти объекты могут выглядеть одинаково и вести себя сходным образом, но, в тоже время, отличаться между собой. Иметь, например, различное местоположение на карте (в игровом мире) или отличаться количеством восстанавливаемого здоровья. В терминологии ООП тип объекта называют классом. Тогда объект – это экземпляр какого-либо класса. Характеристики объекта называют свойствами (реже полями).
Внутри каждого объекта сохраним идентификатор, позволяющий установить, к какому классу он относится. В примере это первое поле структуры. Другие поля структуры будут хранить свойства аптечки, это координаты x:y и количество здоровья, которое она восстановит при использовании. Описание аптечки, которая находится в точке [10;15] и восстанавливает 50 пунктов здоровья:

```
Asm:
dd TYPE_MEDIKIT ;класс - аптечки
dd 10   ;координата x = 10
dd 15   ;           y = 15
dd 50   ;пункты здоровья = 50

C:
struct MEDIKIT {
 int type;
 int x, y;
 int hp;
};
MEDIKIT mk = {TYPE_MEDIKIT, 10, 15, 50};
```

Определим теперь другой объект – бомбу (mine). Она также должна иметь координаты на карте. Но вместо количества здоровья понадобится количество урона, которое она нанесет. Кроме того, бомба имеет еще одну характеристику – время, через которое она сработает после активации. Объект может выглядеть следующим образом:

```
Asm:
dd TYPE_MINE  ;тип объекта = mine
dd 10   ;x = 10
dd 15   ;y = 15
dd 5        ;время срабатывания = 5 секунд
dd 50   ;урон = 50

C:
```

```
struct MINE {
 int type;
 int x, y;
 int timeout;
 int hp;
};
MEDIKIT mk = {TYPE_MINE, 10, 15, 5, 50};
```

Методы (Methods)

Когда игрок встретит в мире один из таких предметов, мы получим сообщение об этом вместе с указателем на структуру данных, описывающих объект. Тогда мы прочитаем первое поле и узнаем, к какому классу относится встреченный объект. Если это аптечка, мы восстанавим здоровье игрока. А если бомба, начнем отсчет до взрыва.

```
Asm:
;esi = адрес структуры данных (объекта)
cmp     dword [esi], TYPE_MEDIKIT
je      touched_medikit
cmp     dword [esi], TYPE_MINE
je      touched_mine

C:
switch(obj->type) {
case TYPE_MEDIKIT:
        MEDIKIT *mk = (MEDIKIT*)obj;
        ...
case TYPE_MINE:
        MINE *mn = (MINE*)obj;
        ...
```

Этот подход прост, но имеет недостатки. Части кода, где происходит обращение к объекту, могут быть разбросаны по всей программе. И если мы хотим изменить что-то в описании класса (изменить структуру данных, описывающую объект), мы должны изменить все места, где происходит обращение к объекту. Это неудобно, вдобавок, можно легко допустить ошибку.

Будет лучше, если мы сгруппируем код. Напишем функции, которые будут работать с объектами и сохраним их в одном месте. В нашем примере есть функция, восстанавливающая здоровье игрока при использовании аптечки, и другая функция, активирующая взрыватель при использовании бомбы.

```
Asm:
cmp     dword [esi], TYPE_MEDIKIT
jne     not_medikit
push    esi
call    medikit_touched
jmp     done
not_medikit:

cmp     dword [esi], TYPE_MINE
jne     not_mine
push    esi
call    mine_touched
jmp     done
not_mine:

C:
switch(obj->type) {
case TYPE_MEDIKIT:
        medikit_touched((MEDIKIT*)obj);
        break;
case TYPE_MINE:
        mine_touched((MINE*)obj);
        break;
```

Функции, которые работают с объектами, называют методами.

Виртуальные методы (Virtual Methods)

Однако, такой подход тоже нельзя назвать идеальным. Если нам понадобится добавить новый класс или новый метод, придется сделать слишком много исправлений. Но мы можем разместить указатель на метод внутри самого объекта. Тогда, вызывая метод мы не будем определять тип объекта, а сразу вызовем функцию:

```
Asm:
; medikit object
dd TYPE_MEDIKIT         ;тип объекта = аптечка
dd medikit_touched      ;указатель на "touched" метод
dd 10                   ;x = 10
dd 15                   ;y = 15
dd 50                   ;HP = 50

; mine object
dd TYPE_MINE            ;тип объекта = бомба
dd mine_touched         ;указатель на "touched" метод
dd 10                   ;x = 10
dd 15                   ;y = 15
dd 5                    ;таймер = 5 секунд
dd 50                   ;урон = 50

; вызов "touched" метода, указатель на объект передается в ESI
    push    esi   ;адрес объекта, для которого вызывается метод
    call    dword [esi+4]   ;вызов метода

C:
struct MEDIKIT {
   int type;
   void (*touched)(OBJECT* this);//указатель на "touched" метод
   int x, y;
   int hp;
};
struct MINE {
   int type;
   void (*touched)(OBJECT* this);//указатель на "touched" метод
   int x, y;
   int timeout;
   int hp;
};
struct OBJECT {  //общая часть, есть во всех объектах
   int type;   //тип объекта
   void (*touched)(OBJECT* this);//указатель на "touched" метод
} *obj;

//вызов метода для объекта
    obj->touched(obj);
```

Инкапсуляция

Здесь мы подошли к основному принципу объектно ориентированного программирования. Согласно нему объект закрыт для других объектов, а взаимодействие осуществляется только через методы. То есть, если другому объекту потребуется узнавать, сколько пунктов здоровья восстанавливает аптечка (чтобы отображать на экране, например) мы напишем еще одну функцию-метод, которая будет возвращать значение свойства hp. Если другому объекту потребуется изменять это свойство, мы также добавим еще один метод. Быть может, это кажется излишней расточительностью, но оно оправдывает себя. Разработав класс мы можем возвращаться к его коду только если хотим что-то добавить (есть специальные средства, чтобы не делать даже этого). А все взаимодействие сводится к вызову однажды разработанных методов, которые не изменяются вместе с измененным классом. Это также позволяет лучше организовать разработку больших программ. Однажды договорившись о взаимодействии классов программисты могут разрабатывать их независимо друг от друга.

Принцип объект-черный ящик называют инкапсуляцией.

Виртуальные таблицы (Virtual Table)

Давайте создадим другой метод, назовем его 'shot', он будет обрабатывать выстрел игрока по объекту. Теперь объекты аптечка и бомба выглядят следующим образом:

```
Asm:
; medikit object
dd TYPE_MEDIKIT
dd medikit_touched
dd medikit_shot     ;мы добавили указатель на новый метод
dd 10
dd 15
dd 50

; mine object
dd TYPE_MINE
dd mine_touched
dd mine_shot    ;указатель на метод mine_shot
dd 10
dd 15
dd 5
dd 50

; вызов "touched" метода для объекта, адрес которого в ESI
push    esi
call    dword [esi+4]

; вызов "shot" метода
push    esi
call    dword [esi+8]

C:
struct MEDIKIT {
        int type;
        void (*touched)(OBJECT* this);
        void (*shot)(OBJECT* this); //мы добавили указатель на         //метод
        int x, y;
        int hp;
};
struct MINE {
        int type;
        void (*touched)(OBJECT* this);
        void (*shot)(OBJECT* this);//указатель на новый метод
        int x, y;
        int timeout;
        int hp;
};
struct OBJECT {       //общая часть
        int type;
        void (*touched)(OBJECT* this);
        void (*shot)(OBJECT* this);
} *obj;

//вызов методов
obj->touched(obj);
obj->shot(obj);
```

Вот примеры этого метода. 'medikit.shot' снижает запас здоровья в аптечке в два раза, а 'mine.shot' взорвет бомбу (таймер устанавливается на ноль).

```
Asm:
medikit_shot:
        mov     esi, [esp+4]
        mov     eax, [esi + MEDIKIT.HP]
        shr     eax, 1
        mov     [esi + MEDIKIT.HP], eax
        retn    4
mine_shot:
        mov     esi, [esp+4]
        mov     dword [esi + MINE.TIMEOUT], 0
        retn    4
C:
```

```
void medikit_shot(MEDIKIT *this)
{
        this->hp = this->hp / 2;
}
void mine_shot(MINE *this)
{
        this->timeout = 0;
}
```

Хранить указатели на методы удобно, но ведь для всех объектов одного класса они одинаковы. Будет лучше, если мы создадим статичные таблицы для каждого класса. Тогда внутри объекта можно будет хранить только указатель на такую таблицу. Кроме того, таблица будет уникальной для класса, а значит, хранить тип объекта тоже не нужно.

```
Asm:
; виртуальная таблица для класса MEDIKIT
medikit_vtab:
        dd medikit_touched
        dd medikit_shot

        ; экземпляр класса MEDIKIT (объект аптечка)
mk:
        dd medikit_vtab       ;указатель на виртуальную таблицу
        dd 10                 ;x
        dd 15                 ;y
        dd 50                 ;hp

;виртуальная таблица для класса MINE
mine_vtab:
        dd mine_touched
        dd mine_shot

;экземпляр класса
mn:
        dd mine_vtab          ;указатель на виртуальную таблицу
        dd 20                 ;x
        dd 20                 ;y
        dd 30                 ;таймер
        dd 50                 ;hp

;вызов "touched" метода для объекта, адрес которого в ESI
push    esi             ;аргумент для метода – адрес объекта
mov     eax, dword [esi] ;EAX = адрес виртуальной таблицы
call    dword [eax]      ;EAX+0 = адрес 'touched' метода

push    esi
mov     eax, dword [esi]
call    dword [eax+4]           ;EAX+4 = адрес 'shot' метода

C:
//описание виртуальной таблицы
struct VTAB {
        void (*touched)(OBJECT* this);
        void (*shot)(OBJECT* this);
};

// аптечка
struct MEDIKIT {
        VTAB *vtab;
        int x,y;
        int hp;
};
VTAB medikit_vtab = { //виртуальная таблица для MEDIKIT
  &medikit_touched,
  &medikit_shot
};
MEDIKIT mk1 = {  //аптечка
  &medikit_vtab, 10, 15, 50
};
MEDIKIT mk2 = {  //вторая аптечка
```

```
  &medikit_vtab, 20, 20, 10
};

// mine
struct MINE {
        VTAB *vtab;
        int x,y;
        int timeout;
        int hp;
};
VTAB mine_vtab = { //виртуальная таблица для MINE
  &mine_touched,
  &mine_shot
};
MINE mn1 = {  //объект MINE
  &mine_vtab, 10, 15, 30, 30
};

//общая часть всех объектов
struct OBJECT {
        VTAB *vtab;
};
OBJECT *obj;

//вызов методов
obj->vtab->touched(obj);
obj->vtab->shot(obj);
```

Заключение

Хотя ООП поддерживается многими языками программирования на уровне архитектуры, сам по себе метод является лишь парадигмой (методологией) программирования; наряду с такими методами, как процедурное программирование. Его назначение - обеспечить более удобную разработку больших проектов.

4. DirectX/OpenGL

Небольшое введение в OpenGL

Автор: Star0, пер. Aquila

Дата: 2002-06-27

Я думаю, вы, как и я, смотрели на эти OpenGL'ные демки, как двигаются по экрану полигоны, меняются различные эффекты и так далее. Также, вполне вероятно, вы не очень сильны в математике и не хотите самостоятельно выводить все эти математические синусоидальные процедуры. OpenGL - это классная библиотека, которая позволит вам создать 3D-вселенную очень быстро, двигать ее и наложить серию спецэффектов, используя простую концепцию API.

В наши дни есть два основных вида программирования под OpenGL, в зависимости от операционной системы, которую вы используете. Обычный путь состоит в использовании glut-библиотеки, чтобы задавать анимацию, которая совместима с linux, win32, sgx-станциями и т.д... Другой путь - эт чистый win32. В последнем случае используются обратный вызов win32-клинта, чтобы переключиться на следующее изображение. Как бы то ни было, результат одинаков. Мы будем рассматривать программирование под win32. Поэтому для использования OpenGL вам сначала нужно инициализровать процедуру окна.

1 Процедура окна

Процедура окна обычно обрабатывает все системные события, которые направляются текущему (отображаемому или нет) окну. Этот принцип очень хорошо работает для GUI и уменьшает количество потребляемых машиной ресурсов.

1.1 Класс окна

Для инициализации процедуры окна вам требуется определить класс, небольшую структуры, в которой задается различная информация, например адрес процедуры окна. Это очень легко сделать:

```
mov     edx,esp                 ; сохраняем стек

push    offset classname
push    0                       ; меню окна
push    0                       ; цвет бэкграунда (черный)
push    0                       ; наш курсор
push    0                       ; наша иконка

push    edx                     ; сохраняем стек
push    0
call    GetModuleHandleA        ; загружаем текущую программу
pop     edx

push    eax                     ; равен hinstance

push    0                       ; дополнительно резервирующаяся память
push    0                       ; дополнительный размер структуры
push    offset Window_proc      ; адрес процедуры окна
push    0                       ; we don't care of style
```

```
        mov     eax,esp

        push    edx
        push    eax
        call    RegisterClassA          ; регистрируем класс
        pop     esp                     ; восстанавливаем стек
```

Как только вы зарегистрировали класс, вам необходимо создать свою процедуру окна

1.2 Создание окна

Это просто: один вызов функции API

```
        push    0

        call    GetModuleHandleA
        push    eax

        push    0
        push    0
        push    480
        push    640
        push    0
        push    0
        push    090000000h                      ; WS_VISIBLE
        push    offset windowname
        push    offset classname
        push    0400000h                        ; WS_EX_CLIENTEDGE

        call    CreateWindowExA
```

Это правильно, но если вы вызовите его таким образом, приложение может повиснуть, почему? Потому что мы еще не создали процедуру окна.

1.3 Процедура окна

Процедура окна получает 4 аргумента. Первый - это хэндл окна. Второй - это сообщение. Третий - это нижний параметр, а четвертый - это верхний параметр.

Так как Windows работает в C-подобной среде, аргументы лежат на стеке. Поэтому к ним легко получить доступ. Также вам потребуется позаботиться о некоторых регистра, не модифицировать ebp, например, или ваша программа будет закрыта.

Ваша процедура окна должна перехватывать некоторые сообщения, которые заданы как win32-константы. Давайте взглянем, на что похожа процедура окна.

```
        lParam          equ     16
        wParam          equ     12
        uMsg            equ     8
        hWnd            equ     4

window_proc:

        cmp     dword ptr [esp+uMsg], 0Fh               ; WM_PAINT
        jne     not_event1

        ... здесь какой-то код по обработке WM_PAINT

not_event1:

        ... здесь вы можете поместить обработку других событий
```

```
not_event99:

        cmp     dword ptr [esp+uMsg], WM_CLOSE
        jne     not_quiting

        push    dword ptr [esp+lParam]
        push    dword ptr [esp+4+wParam]
        push    dword ptr [esp+4+4+uMsg]
        push    dword ptr [esp+4+4+4+hWnd]
        call    DefWindowProc

        ret     16
```

1.4 Обработка события

В силу определенных причин некоторые сообщения принимаются, а некоторые посылаются. Поэтому вам нужно, чтобы текущить тред принимал сообщения, что стоит довольно мало ресурсов. Поэтому не паникуйте. Этот код достаточно общий, он может меняться от программиста к программиста, но делает он примерно одно и то же.

```
        mov     edx,esp
        sub     esp,44
        mov     ebx,esp

        push    ebx
        push    eax

loopit:

        push    ebx

        push    0
        push    0
        push    eax
        push    ebx
        call    GetMessageA

        pop     ebx

        cmp     eax,0
        je      goodbye

        push    ebx
        push    ebx
        call    TranslateMessage
        call    DispatchMessageA

        pop     eax
        pop     ebx

        push    ebx
        push    eax
        jmp     loopit

goodbye:

program_error:

        add     esp,44+4+4

        push    0
        call    ExitProcess
```

Как только ваша процедура окна инициализирована, вам требуется синхронизировать ее с OpenGL.

2 OpenGL

Сейчас у нас есть зарегистрированный класс, созданное окно и маленькая оконная процедура. Но теперь нам нужна помощь OpenGL, чтобы инициализировать систему, графику и множество других вещей. Нам необходимо будет делать их в два момента времени: при создании окна и при обновлении экрана.

2.1 При создании окна

Во-первых, вам нужно инициализировать пиксельный формат, который будет использоваться как двойной графический буфер. Для этого вам необходимо использовать PIXELFORMATDESCRIPTOR, это длинная и скучная структура, в которой нужно заполнить тольконесколько полей

```
mov     ecx,40
sub     esp,ecx
mov     edi,esp
push    edi

xor     eax,eax
repz    stosb

pop     edi

mov     word ptr [edi], 40              ; размер структуры
mov     word ptr [edi+2], 1             ; заполняем версию
mov     dword ptr [edi+4], 37           ; PFD_SUPPORT_OPENGL
                                        ; PFD_DRAW_TO_WINDOW
                                        ; PFD_DOUBLE_BUFFER
mov     byte ptr [edi+9], 16            ; биты цвета
mov     byte ptr [edi+17], 16           ; биты глубины

push    edi
push    dword ptr [currentDC]           ; загружаем DC
call    ChoosePixelFormat
```

Если результат равен 0, вызов API не удавлся, и пользователь не может отобразить разрешение. В противном случае вы можете напрямую установить пиксельный формат.

```
push    edi
push    eax
push    dword ptr [currentDC]           ; загружаем DC
call    SetPixelFormat
```

тогда это равно пиксельному формату. Получить текущий DC очень просто:

```
push    dword ptr [esp+hWnd]
call    GetDC
```

и eax равен текущему DC. Нам также требуется создать контекст OpenGL из этого DC.

```
push    eax
call    wglCreateContext
```

Если eax == 0, то вызов не удался и лучшее, что мы можем сделать - это послать WM_CLOSE, чтобы благополучно закрыть приложение.

Далее мы синхронизируем оба контекста:

```
push    eax             ; контекст OpenGL
```

```
        push    ebx             ; текущий DC
        call    wglMakeCurrent
```

Сделано, OpenGL был проинициализирован, теперь вы можете наложить ряд эффектов, но прежде, чем сделать это, вы должны из разрешить. Делается это так:

```
        push    0B57h           ; CL_COLOR_MATERIAL
        call    glEnable        ; damn easy isn't ?
```

Вы также можете попробовать другие значения: GL_DEPTH, G_LIGHTING.

2.2 Как только был изменен размер окна

Когда Windows закончила с инициализацией, она шлет сообщение WM_SIZE, которое озанчает, что настало время пофиксить все, связанное с графикой. Разрешение экрана посылается в lParam. Поэтому сначала устанавливаем порт просмотра.

```
        mov     ecx,dword ptr [esp+lParam]
        movzx   edx,cx
        shr     ecx,16

        push    ecx
        push    edx                             ; длина
        push    0
        push    0                               ; начало
        call    glViewport                      ; определяем порт просмотра
```

После этого нам нужно определить модель матрицы, я не буду давать урок математики о матрицах и трансформациях, но как обычно это очень легко:

```
        push    1701h           ; GL_PROJECTION
        call    glMatrixMode
```

После этого мы вызываем функцию glLoadIndentity:

```
        call    glLoadIdentity
```

Затем мы вызываем библиотеку GL-эффектов, чтобы добавить перспективу порту просмотра. Эта функция принимает только числа двойной точности. Компилер может инициализировать такое число так:

```
double1         dq      1.0f
```

А вот макрос, облегчающий загонку таких чисел в стек:

```
pushdl  macro   double1

        fld     qword ptr [&double1&]
        sub     esp,8
        fstp    qword ptr [esp]

endm
```

Теперь передадим необходимые параметры порту просмотра.

```
        pushdl  farclip
        pushdl  nearclip
        pushdl  XYration
        pushdl  FovAngle

        call    gluPerpective
```

Перспектива - это необходимый аспект нашего 3D-мира. Перспектива бывает разных видов, например как при обзоре через камеру наблюдения или обзор сцены в формате 16/9.

И, опционально, вы можете установить модель теней для сложной 3D-сцены, установите ее в GL_F:AT, это может ускорить рендеринг изображения.

```
push    01D01h
call    glShadeModel
```

Вот и все с этим.

3 Пришло время показать ваши умения демокодера

Windows говорит процедуре окна, что настало время обновить текущее изображение с помощью сообщения WM_PAINT.

Первое, что нужно сделать - это обновить текущее изображение с помощью сообщения WM_PAINT.

```
push    04100h                  ; GL_COLOR_BUFFER_BIT или
call    glClear                 ; GL_DEPTH_BUFFER_BIT

push    GL_MODEL_VIEW
call    glMatrixMode

call    glLoadIdentity
```

Теперь мы можем установить порт просмотра в виде камеры. У этой камеры есть 9 операндов.

```
3 первых - это начало камеры
3 следующих - это назанчение камеры
3 последних - это координаты верхнего вектора камеры, ее вращение
```

Эти операнды также двойной точности, поэтому используйте pushdl, чтобы использовать эту функцию.

```
pushf...
pushf...
call    gluLookAt
```

А теперь надо отрисовать окружение. Можно сделать много вещей: установить свет, тени, прозрачность, искажения и так далее.

Давайте сделаем такую простую вещь:

```
push    GL_POINTS
call    glBegin

push    0.5
push    0.5
push    0.5
call    glVertex3f

call    glEnd
```

glColor3d и glVertex3i существуют во многих форматах. Последняя буква задает тип переменной, который вы хотите использовать. glVertex3d означается, что вы используете аргумент двойной точности.

Когда окружение построено, вы можете закончить это так:

```
push    DC
call    SwapBuffers
```

И буфер окажется на экране.

4 Небольшое приложение

Часто люди удивляются, почему их продукция отображается на экране очень странно. Причина проста: OpenGL не интересуется другими элементами, расположенными на экране, поэтому н может перерисовать их... Вы можете установить тест глубины во время создания окна:

```
push    0B71h           ; DEPTH_TEST
call    glEnable        ;
```

Некоторые люди хотят полноэкранное разрешение, это можно сделать в два шага, меняя разрешение дисплея:

```
mov     ecx,148
mov     esp,ecx
mov     edi,esp
xor     eax,eax

repz    stosb

mov     dword ptr [esp+36],148          ; dmSize
mov     dword ptr [esp+104],16          ; dmBitsPerPixel
mov     dword ptr [esp+108],640         ; dmWidth
mov     dword ptr [esp+112], 480        ; dmHeight
mov     dword ptr [esp+40], 1C0000h
; DM_BITSPERPEL DM_PELSWIDTH DM_PELSHEIGHT

mov     edx,esp

push    4                       ; CDS_FULLSCREEN
push    edx                     ; dmScreenSettings
call    ChangeDisplay

add     esp,148
```

Не правда ли, это очень легко?

Программирование игр на ассемблере (Часть 1)

Автор: Chris Hobbs, пер. UniSoft

Дата: 2002-10-22

Введение

Я уверен, что многие из вас, уже задумывались о создании игр.
Многие из вас провели немало часов в интернете, в поисках какой-либо информации о том, как писать игры на чистом ассемблере. Но, к сожалению, такой информации почти нет, а тем более на русском.
Я думаю, что этот туториал пополнит ряды русскоязычной документации и надеюсь, что он вам понравится.

О чем этот туториал ?

Этот туториал посвящен разработке игры полностью на ассемблере.
В нем будет рассмотрен широкий круг вопросов - от структурированного кода до графических аспектов.

Для кого это ?

В широком смысле - для тех, кто хотел бы научиться тому, чего возможно не знал прежде...
Да и вообще мало кто знает, что полностью виндовское приложение можно написать на чистом ассемблере.

Что для этого надо ?

Единственное требование - это способность читать.
Однако, если вы желаете писать и транслировать исходный код, то вам понадобится компилятор MASM 6.12+. Вы также можете скачать пакет MASM32, в котором есть все, что вам понадобится.

MASM32v7 можно взять здесь:

- http://www.movsd.com/ или
- http://wasm.ru/tools/7/masm32v7.zip или
- http://spiff.tripnet.se/~iczelion/files/masm32v7.zip.

Почему Ассемблер?

Общеизвестно, что любой компилятор генерит код, в котором неизбежно присутствуют баги.
Ассемблер - трудный язык как для понимания, так и для написания, что особенно справедливо для DOS'a.
С приходом Windows многое меняется.

В чтении труден не только ассемблер, даже в
Си бывают такие дебри, что сам черт ногу сломает.
Читабельность исходников зависит от квалификации программиста и умения комментировать код.
Вообще-то тут еще надо разобраться, что сложнее: сложить 2 переменные на ассемблере или проследить иерархию какой-нибудь виртуальной функции? Здесь комментарий решает все.

Помните: то, что известно вам, не значит, что это же известно другим.

Далее, проблема в переносимости. Предоставленный язык ассемблера не переносим на другие платформы.
Есть, конечно, способ обойти это, который позволяет вам писать код для любой x86 платформы, но его описание не входит в рамки этого туториала.
Большинство игрушек пишутся под винды. Это означает, что код привязан к DirectX и к WIN32API, следовательно вы не сможете переносить код, во всяком случае, без некоторой доработки.

Конечно, ассемблер труден в понимании. На этой странице будут показаны его основы.
Писать ассемблерный код под Windows, особенно с MASM, очень просто.
Это подобно написанию некоторого кода на C. Попытайтесь, и я уверен, что вы не будете разочарованы.

Основы Win32 ASM

Если Вы уже знакомы с языком ассемблера под windows, то можете пропустить этот раздел.
Он является важным дополнением. Для его обсуждения, предполагается,
что вы по крайней мере знакомы с x86 архитектурой.

Первое, что вам нужно понять, это команды (инструкции).

• **MOV**

Эта инструкция копирует значение из одного места в другое.
Вы можете копировать только из регистра в регистр, из памяти в регистр или из регистра в память.
Но с помощью этой инструкции, вы не можете копировать напрямую из памяти в память.

```
Пример:

 MOV EAX, 30
 MOV EBX, EAX
 MOV my_var1, EAX
 MOV DWORD PTR my_var, EAX
```

В первой строке число 30 копируется в регистр EAX. Во второй строке регистр EAX копируется в регистр EBX.
В 3-ей строке регистр EAX копируется в память. В 4-ой строке регистр EAX копируется в то место памяти,
куда УКАЗЫВАЕТ пойнтер my_var , причем префикс DWORD указывает на то, что пересылаются 4 байта.

• **ADD & SUB**

Эти две инструкции выполняют сложение и вычитание, соответственно.

```
Пример:
 ADD EAX, 30
 SUB EBX, EAX
```

Прибавляем 30 к содержимому регистра EAX,
а затем вычитаем полученное значение из регистра EBX.

• **MUL & DIV**

Эти две инструкции выполняют умножение и деление, соответственно.

```
Пример:
 MOV EAX, 10
 MOV ECX, 30
 MUL ECX
 XOR EDX, EDX
 MOV ECX, 10
 DIV ECX
```

Рассмотрим пример: сначала загружаем в EAX=10 и в ECX=30.
EAX по умолчанию всегда один из 2-х сомножителей,
посему третья команда (в ней вы явно указываете второй сомножитель) перемножает EAX и ECX.
Результат от умножения (произведение) будет находится в EAX:EDX.
Перед выполнением деления, сначала Вы должны очистить регистр EDX, что и делает команда XOR,
выполняя исключающее ИЛИ с самим собой. После деления, целая часть результата будет находится в EAX,
а остаток (если есть) в EDX.

Вообще инструкций очень много, но этих пока достаточно для начала.
Вероятно мы будем использовать и некоторые другие, но их не сложно понять,
если только вы поняли основные. Теперь нам нужно разобраться с правилами вызова функций.
Мы будем использовать стандартные правила, такие же как Win32 API. Что это значит, а то,
что мы помещаем параметры в стек справа налево и нам также не надо будет заботиться
об очистке стека от параметров, все это будет для нас прозрачно.
Для вызова функции мы будем использовать псевдо-оператор INVOKE.

Далее, есть одна проблема с вызовом функций Windows.
Чтобы использовать invoke, вы должны иметь прототип функции.

Вообще, MASM обеспечивает высокий уровень синтаксиса при написании.
В нем есть конструкции, позволяющие писать логику If-Then-Else и циклы For loops
аналогично сишным конструкциям.

Разработка

А теперь пришло время для проектирования игры. Этим процессом часто пренебрегают и сразу кидаются
кодировать то, что пришло в голову. Хотя такой подход иногда и проходит,
но чаще всего проваливается.

Как говорится, сначала было слово. Для написания игры нужно представить в голове,
как она будет работать. Иногда просто полезно описать СЛОВАМИ то, что вы хотите видеть в своей игре.

Простая часть закончена, теперь нужно продумать все детали. Будут ли в игре элементы соревнования?
Нужны ли опции сохранения/загрузки? Сколько уровней? Что должно происходить в конце уровня?
Имеется ли вводный экран? И еще много, много вопросов.

После этого нужно будет приниматься за наброски уровней. Какой должен быть экран и интерфейс?
Это не обязательно должно быть точно так, но это даст реальную картину,
как должна выглядеть финальная версия игры.

Код

Всегда имеет смысл насыщать код возможно большим количеством комментариев, чем лично я, например,
всегда пренебрегаю.

Далее следует примерная схема организации ассемблерного игрового кода:

```
;#########################################################################
;#########################################################################
; ABOUT SPACE-TRIS:
;
;   Главная секция - WinMain и т.д.
;
;           - WinMain()
;           - WndProc()
;           - Main_Loop()
;           - Game_Init()
;           - Game_Main()
;           - Game_Shutdown()
;
;#########################################################################
;#########################################################################

;#########################################################################
;#########################################################################
; THE COMPILER OPTIONS
;#########################################################################
;#########################################################################

        .386
        .MODEL flat, stdcall
        OPTION CASEMAP :none   ; case sensitive

;#########################################################################
;#########################################################################
; THE INCLUDES SECTION
;#########################################################################
;#########################################################################

        ;=================================================
        ; хидеры для Windows structs,
        ; unions, constants
        ;=================================================
        INCLUDE Includes\Windows.inc

        ;=================================================
        ; хидеры для Window calls
        ;=================================================
        INCLUDE \masm32\include\comctl32.inc
        INCLUDE \masm32\include\comdlg32.inc
        INCLUDE \masm32\include\shell32.inc
        INCLUDE \masm32\include\user32.inc
        INCLUDE \masm32\include\kernel32.inc
```

```
        INCLUDE \masm32\include\gdi32.inc

        ;====================================
        ; хидеры Direct Draw
        ;====================================
        INCLUDE Includes\DDraw.inc

        ;================================================
        ; либы
        ;================================================
        INCLUDELIB \masm32\lib\comctl32.lib
        INCLUDELIB \masm32\lib\comdlg32.lib
        INCLUDELIB \masm32\lib\shell32.lib
        INCLUDELIB \masm32\lib\gdi32.lib
        INCLUDELIB \masm32\lib\user32.lib
        INCLUDELIB \masm32\lib\kernel32.lib

        ;================================================
        ; хидеры с прототипами
        ;================================================
        INCLUDE Protos.inc

;#########################################################################
;#########################################################################
; LOCAL MACROS
;#########################################################################
;#########################################################################

        szText MACRO Name, Text:VARARG
                LOCAL lbl
                JMP lbl
                Name DB Text,0
                lbl:
        ENDM

        m2m MACRO M1, M2
                PUSH             M2
                POP               M1
        ENDM

        return MACRO arg
                MOV      EAX, arg
                RET
        ENDM

        RGB MACRO red, green, blue
                XOR      EAX,EAX
                MOV      AH,blue
                SHL      EAX,8
                MOV      AH,green
                MOV      AL,red
        ENDM

        hWrite MACRO handle, buffer, size
                MOV      EDI, handle
                ADD      EDI, Dest_index
                MOV      ECX, 0
                MOV      CX, size
                ADD      Dest_index, ECX
                MOV      ESI, buffer
                movsb
        ENDM

        hRead MACRO handle, buffer, size
                MOV      EDI, handle
                ADD      EDI, Spot
                MOV      ECX, 0
                MOV      CX, size
                ADD      Spot, ECX
                MOV      ESI, buffer
                movsb
```

```
        ENDM

;#########################################################################################
;#########################################################################################
;  глобальные переменные
;#########################################################################################
;#########################################################################################

;#########################################################################################
;#########################################################################################
; External variables
;#########################################################################################
;#########################################################################################

;#########################################################################################
;#########################################################################################
; BEGIN INITIALIZED DATA
;#########################################################################################
;#########################################################################################

    .DATA

;#########################################################################################
;#########################################################################################
; BEGIN CONSTANTS
;#########################################################################################
;#########################################################################################

;#########################################################################################
;#########################################################################################
; BEGIN EQUATES
;#########################################################################################
;#########################################################################################

        ;=================
        ;Utility Equates
        ;=================
FALSE           EQU  0
TRUE            EQU  1


;#########################################################################################
;#########################################################################################
; BEGIN THE CODE SECTION
;#########################################################################################
;#########################################################################################

  .CODE

start:

;#########################################################################
; WinMain Function
;#########################################################################

;#########################################################################
; End of WinMain Procedure
;#########################################################################

;#########################################################################
; Main Window Callback Procedure -- WndProc
;#########################################################################

;#########################################################################
; End of Main Windows Callback Procedure
;#########################################################################

;=========================================================================
;=========================================================================
; THE GAME PROCEDURES
;=========================================================================
```

```
;=========================================================================

;#########################################################################
; Game_Init Procedure
;#########################################################################

;#########################################################################
; END Game_Init
;#########################################################################

;#########################################################################
; Game_Main Procedure
;#########################################################################

;#########################################################################
; END Game_Main
;#########################################################################

;#########################################################################
; Game_Shutdown Procedure
;#########################################################################

;#########################################################################
; END Game_Shutdown
;#########################################################################

;#####################################
; THIS IS THE END OF THE PROGRAM CODE #
;#####################################
END start
```

Вот мы и подошли к концу первой статьи.
А теперь хорошая новость - весь скучный материал в основном позади.
Есть еще и плохая новость - вы не видели код, который будет в следующей статье. :)

Программирование игр на ассемблере (Часть 2)

Автор: Chris Hobbs, пер. UniSoft

Дата: 2002-10-29

• На чем мы остановились?
• Синтаксис MASM
• Основной игровой цикл
• Связь с Direct Draw
• Наша Direct Draw Library
• Наша Bitmap Library
• A Game ...
• В следующей статье...

-- > На чем мы остановились?

В прошлой статье были рассмотрены основы Win32 ASM программирования, основы создания игр, и сам процесс разработки. Пришло время зайти немного дальше. Сначала я расскажу о высокоуровневых конструкциях MASM, которые являются удобоваримыми в сравнительном смысле с аналогичными конструкциями на Си. Затем рассмотрим основной цикл и главные оконные процедуры. После чего обратим внимание на Direct Draw и вызовы связанные с ним. Поняв, как это работает, мы сможем построить свою собственную Direct Draw Library, после чего построим свою bitmap file library и в конце напишем программу, которая отображает экран 'Loading Game' и выходит из нее по нажатию клавиши Esc.

Для компиляции вам потребуется пакет MASM32, или по крайней мере MASM 6.11+.[*Можно взять здесьhttp://wasm.ru/tools/7/masm32v7.zip и здесь http://wasm.ru/tools/7/masm615.zip- прим. ред.*]

-- > Синтаксис MASM [*рекомендуем обратиться к источникам наhttp://www.wasm.ru, в частности, посмотреть туториалы Iczelion'a и др.- прим. ред.*]

Досовские варианты ассемблерных листингов представляют в своем большинстве собрание весьма неудобоваримых сочинений, порой непонятных даже квалифицированным программистам. Очень много меток, jmp-ов и прочей нечисти. Но asm не стоит на месте, и в MASM 6.0 макро-конструкции становятся неотъемлемым инструментом разработки.

MASM ныне - такой же легкий в чтении язык, что и Си. Это, конечно, только мое мнение. Давайте теперь рассмотрим некоторые Си-шные конструкции и их аналоги в MASM.

• IF - ELSE IF - ELSE

```
The C version:if ( var1 == var2 ) | The MASM version:.if ( var1 == var2 )
{                                 |     ; Code goes here
    // Code goes here             | .elseif ( var1 == var3 )
}                                 |     ; Code goes here
else                              | .else
if ( var1 == var3 )               |     ; Code goes here
{                                 | .endif
    // Code goes here             |
}                                 |
else                              |
{                                 |
    // Code goes here             |
}                                 |
```

• DO - WHILE

```
The C version:do        | The MASM version:.repeat
{                       |     ; Code goes here
    // Code goes here   | .until ( var1 != var2 )
```

```
}                       |
while ( var1 == var2 ); |
```

• **WHILE**

```
The C version:while ( var1 == var2 ) | The MASM version:.while ( var1 == var2 )
{                                    |     ; Code goes here
    // Code goes here                | .endw
}                                    |
```

Это все - примеры рабочих конструкций, и как вы видите они чрезвычайно просты. При компиляции, MASM скомпилирует в коде, все те же нужные метки, jmp-ы, cmp-конструкции.

Есть еще и другие вещи, которые нам следует обсудить, это псевдо-операторы которые позволяют нам легко определить процедуры/функции, такие как PROTO и PROC. Использовать их очень легко. Для начала, также как и в Си, вам нужно иметь прототип. В MASM это делается с помощью ключевого слова PROTO. Вот несколько примеров объявления прототипов для ваших функций:

```
 ;=================================
 ; Main Program Procedures
 ;=================================
 WinMain PROTO    :DWORD,:DWORD,:DWORD,:DWORD
 WndProc PROTO    :DWORD,:DWORD,:DWORD,:DWORD
```

Вышеупомянутый код сообщает ассемблеру, о двух новых процедурах WinMain и WndProc. Каждая из них имеет список параметров связанных с ними. В каждой функции задаются 4 параметра (DWORD). Для тех кто использует пакет MASM32, то в нем уже есть прототипы всех Windows API функций, вам просто нужно подключить соответствующий файл. Но вам надо убедиться, что все пользовательские процедуры определены вышеупомянутым способом.

Породив прототип, мы можем породить и саму функцию с помощью ключевого слова PROC:

```
;#########################################################################
; WinMain Function
;#########################################################################
WinMain PROC hInstance :DWORD,
  hPrevInst :DWORD,
  CmdLine  :DWORD,
  CmdShow  :DWORD


 ;===========================
 ; We are through
 ;===========================
 return msg.wParam

WinMain endp
;#########################################################################
; End of WinMain Procedure
;#########################################################################
```

При такой записи имеется доступ ко всем параметрам функции. Не правда ли - слишком просто для Winmain?

-- > Основной игровой цикл

Теперь, когда мы все знаем, как писать на ассемблере, давайте приступим к написанию основного игрового цикла.

Начнем с WinMain().

```
   .CODE

start:
 ;=================================
 ; Получим экземпляр
 ; приложения
 ;=================================
```

```
INVOKE GetModuleHandle, NULL
MOV hInst, EAX

;=================================
; Как насчет командной строки?
;=================================
INVOKE GetCommandLine
MOV CommandLine, EAX

;=================================
; Вызов WinMain
;=================================
INVOKE WinMain,hInst,NULL,CommandLine,SW_SHOWDEFAULT

;=================================
; Выход
;=================================
INVOKE ExitProcess,EAX
```

Единственное, что здесь может оказаться немного странным [*можно подумать, что данный случай - исключение, а не правило. Это не странно, а естественно - прим. ред.*], так это пересылка регистра EAX в переменную (MOV ...,EAX) в конце INVOKE. Причина в том, что все функции Windows (и Си функции в том числе), возвращают значение результата функции/процедуры в регистре EAX. Этот код вы можете использовать во всех программах, которые будете писать, по крайней мере, мне никогда не приходилось его изменять.

А теперь сам код:

```
;#########################################################################
; WinMain Function
;#########################################################################
WinMain PROC hInstance :DWORD,
  hPrevInst :DWORD,
  CmdLine  :DWORD,
  CmdShow  :DWORD

 ;========================
 ; локальные переменные (размещаются в стеке)
 ;========================
 LOCAL wc  :WNDCLASS

 ;==================================================
 ; Заполнение структуры WNDCLASS требуемыми переменными
 ;==================================================
 MOV wc.style, CS_OWNDC
 MOV wc.lpfnWndProc,OFFSET WndProc
 MOV wc.cbClsExtra,NULL
 MOV wc.cbWndExtra,NULL
 m2m wc.hInstance,hInst     ;<< замечание: это макрос
 INVOKE GetStockObject, BLACK_BRUSH
 MOV wc.hbrBackground, EAX
 MOV wc.lpszMenuName,NULL
 MOV wc.lpszClassName,OFFSET szClassName
 INVOKE LoadIcon, hInst, IDI_ICON ; icon ID
 MOV wc.hIcon,EAX
 INVOKE LoadCursor,NULL,IDC_ARROW
 MOV wc.hCursor,EAX

 ;================================
 ; Регистрация класса, который мы создали
 ;================================
 INVOKE RegisterClass, ADDR wc

 ;========================================
 ; Создание главного экрана
 ;========================================
 INVOKE CreateWindowEx,NULL,
   ADDR szClassName,
                       ADDR szDisplayName,
```

```
                        WS_POPUP OR WS_CLIPSIBLINGS OR \
    WS_MAXIMIZE OR WS_CLIPCHILDREN,
                        0,0,640,480,
                        NULL,NULL,
                        hInst,NULL

  ;=========================================
  ; сохранить указатель на окно (хэндл)
  ;=========================================
    MOV hMainWnd, EAX

  ;====================================
  ; Скрыть курсор
  ;====================================
  INVOKE ShowCursor, FALSE

  ;=========================================
  ; Вывод на экран нашего окна, которое мы создали
  ;=========================================
  INVOKE ShowWindow, hMainWnd, SW_SHOWDEFAULT

  ;================================
  ; Инициализация игры
  ;================================
  INVOKE Game_Init

  ;=======================================
  ; проверка на ошибки и выход в случае чего
  ;=======================================
  .IF EAX != TRUE
   JMP shutdown
  .ENDIF

  ;====================================
  ; Цикл, пока не будет послано PostQuitMessage
  ;====================================
    .WHILE TRUE
   INVOKE PeekMessage, ADDR msg, NULL, 0, 0, PM_REMOVE
   .IF (EAX != 0)
    ;===================================
    ; Выход из цикла
    ;===================================
    MOV EAX, msg.message
    .IF EAX == WM_QUIT
     ;=====================
     ; Выход
     ;=====================
     JMP shutdown
    .ENDIF

    ;===================================
    ; Translate and Dispatch the message
    ;===================================
    INVOKE TranslateMessage, ADDR msg
    INVOKE DispatchMessage, ADDR msg

   .ENDIF

   ;================================
   ; вызов главного игрового цикла
   ;================================
   INVOKE Game_Main

  .ENDW

shutdown:

  ;================================
  ; Завершение игры
  ;================================
  INVOKE Game_Shutdown
```

```
 ;================================
 ; Показать курсор
 ;================================
 INVOKE ShowCursor, TRUE

getout:
 ;============================
 ; Завершение
 ;============================
 return msg.wParam

WinMain endp
;########################################################################
; End of WinMain Procedure
;########################################################################
```

Давайте проанализируем. Обратите внимание: при инициализации локальной переменной (в нашем случае это структура WNDCLASS), в начале функции не нужно никакой возни со стеком push/pop, также как и в ее конце, за вас все сделает компилятор. Вы должны только объявить локальные переменные, как в Си. Далее, заполняем структуру значениями. Обратите внимание на использование макроса m2m. Это потому, что ASM не позволяет напрямую копировать из памяти в память, без использования регистра или стека в качестве посредника.

Далее, создаем окно и прячем курсор, так как он нам в игре не нужен. Показываем окно и вызываем Game_Init(). Если внутри процедуры Game_Init() произошла ошибка, то ее результат будет FALSE. Проверяем результат процедуры Game_Init() и в случае ошибки выходим из программы (прыгаем на метку shutdown). Вообще, при отладке asm-процедур всегда нужно помнить, что должна быть одна точка входа и одна точка выхода.

Дальше идет цикл сообщений (message loop), которые могут поступать откуда угодно. Если бы это была обычная программа, а не игра, то мы бы использовали GetMessage() для приема сообщений из очереди. Но здесь есть одна проблема, если нет никаких сообщений, то функция будет ждать, пока придет какое-нибудь сообщение. Это совершенно не подходит для игры. Нам нужно постоянно выполнять главный игровой цикл, независимо от того, придут ли какие-либо сообщения или нет. Есть один путь обойти это - использовать PeekMessage(). PeekMessage() возвращает ноль, если нет никаких сообщений, иначе вернет сообщение из очереди.

Обратите внимание, что главный игровой цикл (Game_Main), будет вызываться всегда, независимо от того, пришло ли какое-либо сообщение или нет. Если бы мы этого не сделали, то Windows мог бы обработать кучу сообщений, в то время, как главный игровой цикл - ни разу.

И в конце, когда мы получаем сообщение quit, мы выходим из цикла и соответственно завершаем программу.

-- > Связь с Direct Draw

Мы не будем рассматривать сам DirectX на уровне асм-а, а рассмотрим его на уровне основных концепций.

Прежде всего, необходимо понять саму концепцию Таблицы Виртуальных Функций. Делается запрос в нее в форме смещения, и получается АДРЕС расположения функции. Т.е. под вызовом функции понимается обращение к таблице, которая УЖЕ существует. Адреса функций имеются в DirectX-библиотеке.

Далее, необходимо определить адрес объекта, для которого вызывается функция. Вычисляем виртуальный адрес и сохраняем в стеке все параметры. Для этой цели существуют различные макросы, в частности, DD4INVOKE еще из 4-го DirextX.

Сначала определяем имя функции, затем имя объекта и параметры:

```
    ;======================================
    ; Создадим primary surface
```

```
    ;=====================================
      DD4INVOKE CreateSurface, lpdd, ADDR ddsd, ADDR lpddsprimary, NULL
```

В этом примере создается поверхность, вызовом функции CreateSurface(), передавая ей в качестве параметров: указатель на объект, адрес структуры Direct Draw Surface Describe (ddsd), адрес переменной для хранения указателя на поверхность, и наконец NULL. Теперь, когда мы увидели, как делать запросы к DirectX, давайте построим небольшую библиотеку.

-- > Наша Direct Draw Library

Нам понадобятся функции для инициализации и выхода из игры, для определения формата пикселя, создания и прорисовки поверхностей, и загрузки в нее битмапа.

Далее код функции инициализации:

```
;#########################################################################
; DD_Init Procedure
;#########################################################################
DD_Init PROC screen_width:DWORD, screen_height:DWORD, screen_bpp:DWORD

        ;=======================================================
        ; Устанавливаем полноэкранный режим
        ;=======================================================

        ;=================================
        ; Локальные переменные
        ;=================================
        LOCAL        lpdd_1          :LPDIRECTDRAW

        ;=============================
        ; Создаем объект
        ;=============================
        INVOKE DirectDrawCreate, 0, ADDR lpdd_1, 0

        ;=============================
        ; Обработка ошибок
        ;=============================
        .IF EAX != DD_OK
                ;======================
                ; Ошибка
                ;======================
                INVOKE MessageBox, hMainWnd, ADDR szNoDD, NULL, MB_OK

                ;======================
                ; Выход
                ;======================
                JMP      err

        .ENDIF

        ;=======================================
        ; Получим DirectDraw 4 object
        ;=======================================
        DDINVOKE QueryInterface, lpdd_1, ADDR IID_IDirectDraw4, ADDR lpdd

        ;=======================================
        ; Получили ???
        ;=======================================
        .IF EAX != DD_OK
                ;============================
                ; Нет
                ;============================
                INVOKE MessageBox, hMainWnd, ADDR szNoDD4, NULL, MB_OK

                ;======================
                ; Выход
                ;======================
                JMP      err

        .ENDIF
```

```
;================================================
; Установка cooperative level
;================================================
DD4INVOKE SetCooperativeLevel, lpdd, hMainWnd, \
        DDSCL_ALLOWMODEX OR DDSCL_FULLSCREEN OR \
        DDSCL_EXCLUSIVE OR DDSCL_ALLOWREBOOT

;========================================
; Получили ???
;========================================
.IF EAX != DD_OK
        ;==============================
        ; Нет
        ;==============================
        INVOKE MessageBox, hMainWnd, ADDR szNoCoop, NULL, MB_OK

        ;======================
        ; Выход
        ;======================
        JMP      err

.ENDIF

;================================================
; Установка Display Mode
;================================================
DD4INVOKE SetDisplayMode, lpdd, screen_width, \
        screen_height, screen_bpp, 0, 0

;========================================
; Установили ???
;========================================
.IF EAX != DD_OK
        ;==============================
        ; Нет
        ;==============================
        INVOKE MessageBox, hMainWnd, ADDR szNoDisplay, NULL, MB_OK

        ;======================
        ; Выход
        ;======================
        JMP      err

.ENDIF

;================================
;  screen info
;================================
m2m     app_width, screen_width
m2m     app_height, screen_height
m2m     app_bpp, screen_bpp

;======================================
; Зададим параметры для поверхности (surface)
;======================================
DDINITSTRUCT OFFSET ddsd, SIZEOF(DDSURFACEDESC2)
MOV      ddsd.dwSize, SIZEOF(DDSURFACEDESC2)
MOV      ddsd.dwFlags, DDSD_CAPS OR DDSD_BACKBUFFERCOUNT;
MOV      ddsd.ddsCaps.dwCaps, DDSCAPS_PRIMARYSURFACE OR \
               DDSCAPS_FLIP OR DDSCAPS_COMPLEX
MOV      ddsd.dwBackBufferCount, 1

;======================================
; Создадим первичную поверхность (primary surface)
;======================================
DD4INVOKE CreateSurface, lpdd, ADDR ddsd, ADDR lpddsprimary, NULL

;========================================
; Создали ???
;========================================
.IF EAX != DD_OK
```

```
                ;==============================
                ; Нет
                ;==============================
                INVOKE MessageBox, hMainWnd, ADDR szNoPrimary, NULL, MB_OK

                ;======================
                ; Выход
                ;======================
                JMP      err

        .ENDIF

        ;========================================
        ; Попробуем получить  backbuffer
        ;========================================
        MOV      ddscaps.dwCaps, DDSCAPS_BACKBUFFER
        DDS4INVOKE GetAttachedSurface, lpddsprimary, ADDR ddscaps, ADDR lpddsback

        ;========================================
        ; Получили ???
        ;========================================
        .IF EAX != DD_OK
                ;==============================
                ; Нет
                ;==============================
                INVOKE MessageBox, hMainWnd, ADDR szNoBackBuffer, NULL, MB_OK

                ;======================
                ; Выход
                ;======================
                JMP      err

        .ENDIF

        ;========================================
        ; Получим RGB format для surface
        ;========================================
        INVOKE DD_Get_RGB_Format, lpddsprimary

done:
        ;==================
        ; Все Ок!  :)
        ;==================
        return TRUE

err:
        ;==================
        ; Ничего не Ок! :(
        ;==================
        return FALSE

DD_Init ENDP
;#########################################################################
; END DD_Init
;#########################################################################
```

Рассмотрим подробнее.

Сначала создаем так называемый default Direct Draw object с помощью функции DirectDrawCreate(). Это не более, чем простой вызов функции с несколькими параметрами. Это еще не виртуальная функция. Поэтому мы можем вызывать ее с помощью INVOKE. Также, обратите внимание, что мы потом проверяем на ошибку. Это очень важно в DirectX!!! В случае ошибки, мы просто выводим сообщение, и переходим на метку err: в конце процедуры.

Далее делаем запрос на получение DirectDraw4 object. После чего устанавливаем режимы экрана с помощью SetCooperativeLevel() и SetDisplayMode(). Не забывайте проверять на ошибки, после каждого вызова функций.

На следующем шаге создаем первичную поверхность (primary surface), и в случае успеха создаем back buffer. При этом полученную структуру надо очистить с помощью макроса DDINITSTRUCT, который я включил в файл Ddraw.inc.

После, вызывается процедура для определения формата пикселя для нашей поверхности.

В следующей процедуре мы получаем формат пикселя:

```
;#########################################################################
; DD_Get_RGB_Format Procedure
;#########################################################################
DD_Get_RGB_Format       PROC surface:DWORD

        ;=======================================================
        ; Установим несколько глобальных переменных
        ;=======================================================

        ;====================================
        ; Локальные переменные
        ;====================================
        LOCAL        shiftcount      :BYTE

        ;================================
        ; получим  surface despriction
        ;================================
        DDINITSTRUCT ADDR ddsd, sizeof(DDSURFACEDESC2)
        MOV      ddsd.dwSize, sizeof(DDSURFACEDESC2)
        MOV      ddsd.dwFlags, DDSD_PIXELFORMAT
        DDS4INVOKE GetSurfaceDesc, surface, ADDR ddsd

        ;==============================
        ; маски
        ;==============================
        m2m     mRed, ddsd.ddpfPixelFormat.dwRBitMask      ; Red Mask
        m2m     mGreen, ddsd.ddpfPixelFormat.dwGBitMask    ; Green Mask
        m2m     mBlue, ddsd.ddpfPixelFormat.dwBBitMask     ; Blue Mask

        ;==================================
        ; определим red mask
        ;==================================
        MOV      shiftcount, 0
        .WHILE (!(ddsd.ddpfPixelFormat.dwRBitMask & 1))
                SHR      ddsd.ddpfPixelFormat.dwRBitMask, 1
                INC      shiftcount
        .ENDW
        MOV      AL, shiftcount
        MOV      pRed, AL

        ;====================================
        ; определим green mask
        ;====================================
        MOV      shiftcount, 0
        .WHILE (!(ddsd.ddpfPixelFormat.dwGBitMask & 1))
                SHR      ddsd.ddpfPixelFormat.dwGBitMask, 1
                INC      shiftcount
        .ENDW
        MOV      AL, shiftcount
        MOV      pGreen, AL

        ;====================================
        ; определим blue mask
        ;====================================
        MOV      shiftcount, 0
        .WHILE (!(ddsd.ddpfPixelFormat.dwBBitMask & 1))
                SHR      ddsd.ddpfPixelFormat.dwBBitMask, 1
                INC      shiftcount
        .ENDW
        MOV      AL, shiftcount
        MOV      pBlue, AL
        ;=========================================
```

```
        ; определим специальную переменную для 16 bit mode
        ;=========================================
        .IF app_bpp == 16
                .IF pRed == 10
                        MOV      Is_555, TRUE
                .ELSE
                        MOV      Is_555, FALSE
                .ENDIF
        .ENDIF

done:
        ;===================
        ; все
        ;===================
        return TRUE

DD_Get_RGB_Format       ENDP
;########################################################################
; END DD_Get_RGB_Format
;########################################################################
```

Сначала, инициализируем структуру description, а затем делаем вызов из Direct Draw для получения surface description. Возвращенные маски мы разместим в глобальных переменных для их дальнейшего использования. Маска это значение, которое мы можем использовать для установки или очистки некоторых бит в переменной или регистре. В нашем случае маски используются для того, чтобы получить доступ к red, green, и blue - битам пикселя.

Следующие три секции кода используется для определения числа битов для каждой цветовой компоненты. Например, если нам нужен цветовой режим 24 bpp, то на каждую компоненту нужно отводить по 8 бит. Делается это путем битового сдвига вправо и операции AND.

В случае установки 16-битного режима, переменная Is_555 становится TRUE для режима 5-5-5, или FALSE для режима 5-6-5.

И еще одна функция - для прорисовки текста. Она использует GDI:

```
;########################################################################
; DD_Draw_Text Procedure
;########################################################################
DD_Draw_Text PROC    surface:DWORD, text:DWORD, num_chars:DWORD,
                        x:DWORD, y:DWORD, color:DWORD

        ;=======================================================
        ; Эта функция будет рисовать текст
        ; с помощью GDI
        ;=======================================================

        ;=========================================
        ; Для начала получим  DC
        ;=========================================
        DDS4INVOKE GetDC, surface, ADDR hDC

        ;=========================================
        ; установим цвет текста
        ;=========================================
        INVOKE SetTextColor, hDC, color
        INVOKE SetBkMode, hDC, TRANSPARENT

        ;=========================================
        ; запишем текст в позицию
        ;=========================================
        INVOKE TextOut, hDC, x, y, text, num_chars

        ;=========================================
        ; release DC
        ;=========================================
        DDS4INVOKE ReleaseDC, surface, hDC

done:
```

```
        ;==================
        ; все
        ;==================
        return TRUE

DD_Draw_Text    ENDP
;#########################################################################
; END DD_Draw_Text
;#########################################################################
```

Далее, получаем контекст устройства (DC) для нашей поверхности - это первая вещь, которую нужно получить при рисовании. Устанавливаем background mode и цвет текста с помощью все той же Windows GDI - и мы готовы рисовать текст с помощью вызова TextOut(). После чего освобождаем DC.

Далее, напишем код для наложения bitmap.

-- > Наша Bitmap Library

Нам нужны 2 процедуры: для загрузки битмапа и его прорисовки. В данном случае рассматривается уникальный формат файла.

Этот формат, вероятно один из самых простых, с которым вы когда-либо столкнетесь. Он состоит из 5 основных частей: Width, Height, BPP, Size of Buffer, Buffer. Первые 3 дают информацию о самом образе. В данном случае применяется режим 16 bpp.

```
;#########################################################################
; Create_From_SFP Procedure
;#########################################################################
Create_From_SFP PROC ptr_BMP:DWORD, sfp_file:DWORD, desired_bpp:DWORD

        ;=======================================================
        ; битмап будет загружаться из SFP file.
        ;=======================================================

        ;=================================
        ; Local Variables
        ;=================================
        LOCAL        hFile           :DWORD
        LOCAL        hSFP            :DWORD
        LOCAL        Img_Left        :DWORD
        LOCAL        Img_Alias       :DWORD
        LOCAL        red             :DWORD
        LOCAL        green           :DWORD
        LOCAL        blue            :DWORD
        LOCAL        Dest_Alias      :DWORD

        ;=================================
        ; Создадим этот SFP file
        ;=================================
        INVOKE CreateFile, sfp_file, GENERIC_READ,FILE_SHARE_READ, \
                NULL,OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,NULL
        MOV      hFile, EAX

        ;=================================
        ;  ошибка
        ;=================================
        .IF EAX == INVALID_HANDLE_VALUE
                JMP err
        .ENDIF

        ;=================================
        ; Получим размер файла
        ;=================================
        INVOKE GetFileSize, hFile, NULL
        PUSH     EAX

        ;=================================
        ; ошибка
```

```
;================================
.IF EAX == -1
        JMP      err
.ENDIF

;============================================
; получим память
;============================================
INVOKE GlobalAlloc, GMEM_FIXED, EAX
MOV      hSFP, EAX

;==================================
;  ошибка
;==================================
.IF EAX == 0
        JMP      err
.ENDIF

;==================================
; положим файл в память
;==================================
POP      EAX
INVOKE ReadFile, hFile, hSFP, EAX, OFFSET Amount_Read, NULL

;==================================
; ошибка
;==================================
.IF EAX == FALSE
        ;========================
        ; failed
        ;========================
        JMP      err

.ENDIF

;==================================
; Определим размер
;==================================
MOV      EBX, hSFP
MOV      EAX, DWORD PTR [EBX]
ADD      EBX, 4
MOV      ECX, DWORD PTR [EBX]
MUL      ECX
PUSH     EAX

;=====================================
; Разберемся с типом буфера
;=====================================
.IF desired_bpp == 16
        ;============================
        ; просто установим 16-bit
        ;============================
        POP      EAX
        SHL      EAX, 1
        INVOKE GlobalAlloc, GMEM_FIXED, EAX
        MOV      EBX, ptr_BMP
        MOV      DWORD PTR [EBX], EAX
        MOV      Dest_Alias, EAX

        ;=====================================
        ;  ошибка
        ;=====================================
        .IF EAX == FALSE
                ;========================
                ; облом
                ;========================
                JMP      err

        .ENDIF

.ELSE
```

```
        ;=======================================
        ; код для 24 bit
        ;=======================================

        ;============================
        ;  ошибка
        ;============================
        JMP      err

.ENDIF

;===================================
; подготовим чтение
;===================================
MOV      EBX, hSFP
ADD      EBX, 10
MOV      EAX, DWORD PTR[EBX]
MOV      Img_Left, EAX
ADD      EBX, 4
MOV      Img_Alias, EBX

;===================================
; конвертация
;===================================
.WHILE Img_Left > 0
        ;=================================
        ; создадим color word
        ;=================================
        .IF desired_bpp == 16
                ;=========================================
                ; прочтем по байту для blue, green , red
                ;=========================================
                XOR      ECX, ECX
                MOV      EBX, Img_Alias
                MOV      CL, BYTE PTR [EBX]
                MOV      blue, ECX
                INC      EBX
                MOV      CL, BYTE PTR [EBX]
                MOV      green, ECX
                INC      EBX
                MOV      CL, BYTE PTR [EBX]
                MOV      red, ECX

                ;======================
                ; Img_Alias
                ;======================
                ADD      Img_Alias, 3

                ;================================
                ;  555 или 565 ?
                ;================================
                .IF Is_555 == TRUE
                        ;============================
                        ;  555
                        ;============================
                        RGB16BIT_555 red, green, blue
                .ELSE
                        ;============================
                        ;  565
                        ;============================
                        RGB16BIT_565 red, green, blue

                .ENDIF

                ;================================
                ; перевод в  buffer
                ;================================
                MOV      EBX, Dest_Alias
                MOV      WORD PTR [EBX], AX

                ;============================
```

```
                ; делим на 2
                ;============================
                ADD      Dest_Alias, 2

        .ELSE
                ;=======================================
                ; код для 24 bit
                ;=======================================

                ;============================
                ;  ошибка
                ;============================
                JMP      err

        .ENDIF

        ;=====================
        ; Sub amount left by 3
        ;=====================
        SUB      Img_Left, 3

    .ENDW

    ;===================================
    ; Почистим память
    ;===================================
    INVOKE GlobalFree, hSFP

done:
    ;===================
    ; Вроде все хорошо
    ;===================
    return TRUE

err:
    ;===================================
    ; Почистим SFP Memory
    ;===================================
    INVOKE GlobalFree, hSFP

    ;===================
    ; Не получилось
    ;===================
    return FALSE

Create_From_SFP ENDP
;#########################################################################
; END Create_From_SFP
;#########################################################################
```

Сначала создаем файл, а потом выделяем для него память и читаем данные. После размещения файла в памяти определяем размер буфера.

Далее функция загрузки. Читаем 3 байта и определяем значение переменной (5-6-5 или 5-5-5) для буфера, после чего сохраняем ее там. Каждый пиксель битмапа конвертируем, для чего используется макрос.

После конвертации мы возвращаем буфер с отконвертированными пикселами.

После загрузки в память битмап можно нарисовать в back buffer:

```
;#########################################################################
; Draw_Bitmap Procedure
;#########################################################################
Draw_Bitmap PROC     surface:DWORD, bmp_buffer:DWORD, lPitch:DWORD, bpp:DWORD

    ;=======================================================
    ; Эта функция рисует BMP .
    ; используются width и height экрана
    ;=======================================================
```

```
        ;============================
        ; Локальные переменные
        ;============================
        LOCAL        dest_addr       :DWORD
        LOCAL        source_addr     :DWORD

        ;============================
        ; инициализация
        ;============================
        MOV      EAX, surface
        MOV      EBX, bmp_buffer
        MOV      dest_addr, EAX
        MOV      source_addr, EBX

        MOV      EDX, 480

        ;================================
        ; 16 bit mode
        ;================================

        copy_loop1:
        ;============================
        ; Setup num of bytes in width
        ; 640*2/4 = 320.
        ;============================
        MOV      ECX, 320

        ;============================
        ; установим source и dest
        ;============================
        MOV      EDI, dest_addr
        MOV      ESI, source_addr

        ;======================================
        ; Move с помощью dwords
        ;======================================
        REP      movsd

        ;==============================
        ;  variables
        ;==============================
        MOV      EAX, lPitch
        MOV      EBX, 1280
        ADD      dest_addr, EAX
        ADD      source_addr, EBX

        ;========================
        ; декремент
        ;========================
        DEC EDX

        ;========================
        ; Конец ?
        ;========================
        JNE copy_loop1

done:
        ;====================
        ; Да
        ;====================
        return TRUE

err:
        ;====================
        ; Нет
        ;====================
        return FALSE

Draw_Bitmap     ENDP
;###########################################################################
; END Draw_Bitmap
```

```
;#########################################################################
```

Общеизвестно, что обращение к регистрам осуществляется намного быстрее, чем к памяти. Поэтому адреса источника и приемника мы размещаем в регистрах.

Затем вычисляем количество WORD-значений, которое делим на 2, и получаем количество DWORD-значений. Вообще, используем 640 x 480 x 16. Число 320 разместим в регистре ECX. Делаем классическое REP MOVSD. Двигаем DWORD-ми, вычитаем из ECX по 1, сравнивая с ZERO, если нет, то MOVE A DWORD, до тех пор пока ECX не станет равен 0. Все это здорово смахивает на Си-шный for со счетчиком в ECX. Повторяем 480 раз - по количеству строк.

Теперь осталось все это вывести на экран.

-- > A Game ...

Итак, функции библиотек мы написали, и теперь готовы приступать к основному коду игры. Начнем с инициализации, так как она выполняется в начале нашей программы:

```
;#########################################################################
; Game_Init Procedure
;#########################################################################
Game_Init       PROC

        ;=======================================================
        ;  setup the game
        ;=======================================================

        ;=============================================
        ; инициализация Direct Draw -- 640, 480, bpp
        ;=============================================
        INVOKE DD_Init, 640, 480, screen_bpp

        ;====================================
        ; ошибка
        ;====================================
        .IF EAX == FALSE
                ;========================
                ; облом
                ;========================
                JMP      err

        .ENDIF

        ;====================================
        ; читаем битмап и создаем буффер
        ;====================================
        INVOKE Create_From_SFP, ADDR ptr_BMP_LOAD, ADDR szLoading, screen_bpp

        ;====================================
        ;  ошибка
        ;====================================
        .IF EAX == FALSE
                ;========================
                ; облом
                ;========================
                JMP      err

        .ENDIF

        ;==================================
        ;  DirectDraw back buffer
        ;==================================
        INVOKE DD_Lock_Surface, lpddsback, ADDR lPitch

        ;============================
        ;  ошибка
        ;============================
        .IF EAX == FALSE
                ;==================
```

```
                ;  облом
                ;===================
                JMP      err

        .ENDIF

        ;===================================
        ; рисуем битмап
        ;===================================
        INVOKE Draw_Bitmap, EAX, ptr_BMP_LOAD, lPitch, screen_bpp

        ;===================================
        ;  back buffer
        ;===================================
        INVOKE DD_Unlock_Surface, lpddsback

        ;============================
        ; ошибка
        ;============================
        .IF EAX == FALSE
                ;===================
                ;  облом
                ;===================
                JMP      err

        .ENDIF

        ;====================================
        ;  loading
        ;====================================
        INVOKE DD_Flip

        ;============================
        ;  ошибка
        ;============================
        .IF EAX == FALSE
                ;===================
                ;  облом
                ;===================
                JMP      err

        .ENDIF

done:
        ;===================
        ; да
        ;===================
        return TRUE

err:
        ;===================
        ; нет
        ;===================
        return FALSE

Game_Init       ENDP
;#########################################################################
; END Game_Init
;#########################################################################
```

Эта функция играет важную роль в нашей игре. В этой функции мы делаем вызов процедуры инициализации Direct Draw и в случае успеха загружаем с диска наш битмап. Далее беремся за back buffer и рисуем в него наш битмап. После чего делаем флиппинг видимого и невидимого буферов.

Далее у нас на пути функция WndProc, которая, как известно, обрабатывает сообщения. Если мы захотим добавить обработку еще какого-либо сообщения, то код его обработчика войдет именно в эту функцию.

```
;#########################################################################
; Main Window Callback Procedure -- WndProc
;#########################################################################
WndProc PROC hWin   :DWORD,
             uMsg   :DWORD,
             wParam :DWORD,
             lParam :DWORD

.IF uMsg == WM_COMMAND
        ;============================
        ; без меню
        ;============================

.ELSEIF uMsg == WM_KEYDOWN
        ;======================================
        ; не будем программировать Direct input
        ;======================================
        MOV      EAX, wParam
        .IF EAX == VK_ESCAPE
                ;============================
                ; закрыть программу
                ;============================
                INVOKE PostQuitMessage,NULL

        .ENDIF

        ;============================
        ; processed it
        ;============================
        return 0

.ELSEIF uMsg == WM_DESTROY
        ;============================
        ; закрыть программу
        ;============================
        INVOKE PostQuitMessage,NULL
        return 0

.ENDIF

;================================================
; procedure handle the message
;================================================
INVOKE DefWindowProc,hWin,uMsg,wParam,lParam

RET

WndProc endp
;#########################################################################
; End of Main Windows Callback Procedure
;#########################################################################
```

Я думаю этот код не требует пояснений. Пока мы имеем дело только с 2-мя сообщениями - WM_KEYDOWN и WM_DESTROY. Мы обрабатываем сообщение WM_KEYDOWN для того, чтобы пользователь мог выйти из игры по нажатию клавиши escape. Обратите внимание, что те сообщения, которые мы не обрабатываем, обрабатываются функцией DefWindowProc(). Эта функция уже определена в Windows. Вы только должны вызвать ее всякий раз, когда не обрабатываете сообщение.

И далее процедура завершения:

```
;#########################################################################
; Game_Shutdown Procedure
;#########################################################################
Game_Shutdown   PROC

        ;============================
        ; Shutdown DirectDraw
        ;============================
```

```
        INVOKE DD_ShutDown

        ;=========================
        ; освобождаем память битмап'а
        ;=========================
        INVOKE GlobalFree, ptr_BMP_LOAD

done:
        ;===================
        ; Завершено успешно
        ;===================
        return TRUE

err:
        ;===================
        ; Мы не завершились
        ;===================
        return FALSE

Game_Shutdown   ENDP
;########################################################################
; END Game_Shutdown
;########################################################################
```

Итак, здесь мы выгружаем Direct Draw library, и освобождаем память, которую выделяли под битмап.

-- > В следующей статье...

В следующей статье мы познакомимся с программированием Direct Input. А также создадим свое меню.

Счастливого кодирования!!!

Программирование игр на ассемблере (Часть 3)

Автор: Chris Hobbs, пер. UniSoft

Дата: 2003-02-22

-- > От переводчика...

По вашим многочисленным просьбам продолжаю перевод этой серии туториалов.
И начнем с того, что же такое **DirectInput** и для чего он нужен? **DirectInput** был создан с целью решить проблемы со скоростью ввода. То есть это компонент, который получает данные ввода от пользователя с различных устройств (клавиатура, мышь, джойстик и т.д.) в обход операционной системы.
Итак, основные преимущества:

• Максимальное быстродействие
• Поддержка дополнительных устройств (например, с обратной связью)
• Получение текущего состояния даже без фокуса ввода
• Получения информации из драйвера в обход настроек операционной системы (например, автоповтор)

-- > Так, и на чем же мы остановились в прошлой статье? Если я правильно помню, то в прошлой статье, мы написали код, который выводит загрузочный экран игры. Это значит, что у нас уже есть **DirectDraw** и **Bitmap** библиотеки, но это еще не все. А для обработки ввода с клавиатуры, вместо **DirectInput**, мы использовали сообщение **WM_KEYDOWN**.

Давайте продолжим. Сначала нам нужно создать **DirectInput library**. После этого напишем несколько процедур для синхронизации. Затем разработаем код для меню. **-- > Direct Input** Вы готовы? Отлично! Код **DirectInput** имеет почти тот же формат, что и код для **DirectDraw**.
Давайте рассмотрим код подпрограммы обработки чтения с клавиатуры. (В прилагаемом исходнике также имеется код для обработки мыши, но так как в нашей игре мы ее не используем, то в этих статьях я не буду затрагивать эту тему.)

Начнем с... думаю стоит начать с процедуры инициализации **DirectInput**.

```
;#########################################################################
; DI_Init Procedure
;#########################################################################
 DI_Init PROC

 ;=======================================================
 ; Эта функция установит Direct Input
 ;=======================================================

 ;============================
 ; Создадим объект  DirectInput
 ;============================
 INVOKE DirectInputCreate, hInst, DIRECTINPUT_VERSION, ADDR lpdi,0

 ;============================
 ; ошибки???
 ;============================
 .IF EAX != DI_OK
  JMP err
 .ENDIF

 ;============================
 ; Инициализируем клавиатуру
 ;============================
 INVOKE DI_Init_Keyboard

 ;============================
```

```
; ошибки???
;============================
.IF EAX == FALSE
 JMP err
.ENDIF

;============================
; Инициализируем мышь
;============================
INVOKE DI_Init_Mouse

;============================
; ошибки???
;============================
.IF EAX == FALSE
 JMP err
.ENDIF

done:
;==================
; Выполнено успешно
;==================
return TRUE

err:
;==================
; Вывести сообщение об ошибке
;==================
INVOKE MessageBox, hMainWnd, ADDR szNoDI, NULL, MB_OK

;==================
; Упс!... Ошибка!
;==================
return FALSE

DI_Init ENDP
;########################################################################
; END DI_Init
;########################################################################
```

Этот код совсем не сложный. И начинается он с создания основного объекта **DirectInput**, вызовом **DirectInputCreate()**. Это объект, который используется для получения всех объектов устройств.
Вот ее описание:

DirectInputCreate(HINSTANCE hInst, DWORD dwVer, LPDIRECTINPUT *lplpDI, LPUNKNOWN pU) ;

• **hInst** - Дескриптор экземпляра приложения или DLL.
• **dwVer** - версия объекта **DirectInput**. Использование **DIRECTINPUT_VERSION** позволяет использовать версию по умолчанию.
• **lplpDI** - указатель на указатель интерфейса, который примет в себя информацию.
• **pU** - указатель наследования или агрегирования **COM**. Нас он не интересует - достаточно передать **NULL**.

Затем вызываем процедуры инициализации клавиатуры и мыши.
А теперь посмотрим на саму процедуру инициализации клавиатуры:

```
;########################################################################
; DI_Init_Keyboard  Procedure
;########################################################################
 DI_Init_Keyboard  PROC

 ;=====================================================
 ; Эта функция инициализирует клавиатуру
 ;=====================================================

 ;============================
 ; Получение интерфейса устройства
 ;============================
```

```
  DIINVOKE CreateDevice, lpdi, ADDR GUID_SysKeyboard, ADDR lpdikey, 0

  ;============================
  ; ошибки???
  ;============================
  .IF EAX != DI_OK
   JMP err
  .ENDIF

  ;=========================
  ; Установим уровень доступа
  ;=========================
  DIDEVINVOKE SetCooperativeLevel, lpdikey, hMainWnd, \
            DISCL_NONEXCLUSIVE OR DISCL_BACKGROUND

  ;============================
  ; ошибки???
  ;============================
  .IF EAX != DI_OK
   JMP err
  .ENDIF

  ;=========================
  ; Установим формат данных
  ;=========================
  DIDEVINVOKE SetDataFormat, lpdikey, ADDR c_dfDIKeyboard

  ;============================
  ; ошибки???
  ;============================
  .IF EAX != DI_OK
   JMP err
  .ENDIF

  ;==================================
  ; Теперь попытаемся захватить устройство
  ;==================================
  DIDEVINVOKE Acquire, lpdikey

  ;============================
  ; ошибки???
  ;============================
  .IF EAX != DI_OK
   JMP err
  .ENDIF

  done:
  ;==================
  ; Успешное завершение
  ;==================
  return TRUE

  err:
  ;==================
  ; Упс!... Ошибка!!! :(
  ;==================
  return FALSE

  DI_Init_Keyboard ENDP
 ;#########################################################################
 ; END DI_Init_Keyboard
 ;#########################################################################
```

Первое, что делает эта процедура, это пытается "создать" устройство, от которого мы хотим получать данные (в нашем случае, это клавиатура, но может быть и мышь, и джойстик, и т.д.). Для его создания воспользуемся функцией **CreateDevice()**:

```
CreateDevice(REFGUID rg, LPDIRECTINPUTDEVICE *lplpDI, LPUNKNOWN pU);
```

• **rg** - уникальный идентификатор устройства. Для стандартных устройств заранее определены значения: **GUID_SysKeyboard** для клавиатуры; **GUID_SysMouse** для мыши. Остальные идентификаторы устройств (например джойстики) можно получить вызовом **EnumDevices()**.
• **lplpDI** - указатель на указатель интерфейса, с помощью которого мы будем впоследствии работать с устройством.
• **pU** - все то же агрегирование.

[**Примечание переводчика**]

Обратите внимание: Согласно описанию функции, она содержит 3 параметра, а мы передаем 4 !!! Это связано с тем, что DIINVOKE это макрос, которому первым параметром мы должны передать указатель на интерфейс DirectInput, да, да, тот самый, который мы получили функцией DirectInputCreate().
Более подробно о технологии **COM** можно почитать здесь... .

Получив указатель на интерфейс устройства, устанавливаем уровень доступа. Несмотря на то, что **DirectInput** сам устанавливает уровень доступа по умолчанию, лучше эту функцию все-таки вызвать, но важно определиться заранее - какой доступ необходим.

Уровни доступа:

• **Эксклюзивный (exclusive)/Совместный (non-exclusive)** - первое чем хорош эксклюзивный режим, это тем, что он "подавляет" большинство системных комбинаций (кроме **Alt+TAB** и **Alt+Ctrl+Del**, кстати говоря - есть и отдельный флажок при установке уровня кооперации запрещающий системные комбинации **Windows**), позволяя обработать нажатие, скажем, **Ctrl+Esc**. Следует помнить, что ни одно приложение не сможет получить эксклюзивного доступа к устройству, если такой доступ уже был получен другим приложением, однако сможет получить совместный.
• **Активный (foreground)/Фоновый (background)** - при фоновом доступе приложение сможет читать данные с устройства даже когда окно приложения не активно. В случае же активного доступа приложение сможет читать данные, только если его окно обладает "фокусом" (активно).

Уровень доступа устройства устанавливается вызовом **SetCooperativeLevel()**. **HRESULT SetCooperativeLevel (HWND hwnd, DWORD dwFlags);**

• **hwnd** - Дескриптор окна связанный с устройством.
• **flags** - Флаг совместного доступа:
• **DISCL_BACKGROUND** - Приложение использует устройство когда имеет фокус ввода
• **DISCL_FOREGROUND** - Приложение может использовать устройство даже когда не активно
• **DISCL_EXCLUSIVE** - Монопольный доступ к устройству
• **DISCL_NONEXCLUSIVE** - Приложение не требует монопольного доступа

Приложение должно обязательно определить **DISCL_FOREGROUND** или **DISCL_BACKGROUND** и **DISCL_EXCLUSIVE** или **DISCL_NONEXCLUSIVE**.

После того как установлен уровень кооперации, обязательно нужно установить формат данных. Смысл установки формата данных в том, чтобы указать какие части устройства (например кнопки мыши) будут использоваться. Функция **SetDataFormat()** получает всего один параметр - указатель на структуру описывающую формат данных для устройства. Для стандартных устройств заранее определены глобальные переменные: **c_dfDIKeyboard**, **c_dfDIMouse**, **c_dfDIMouse2**, **c_dfDIJoystick**, **c_dfDIJoystick2**. Итак - формат данных установлен - **DirectInput** знает тип устройства.

И последнее - как и в случае с **DirectDraw**, когда приложение могло "потерять" память поверхностей - приложение **DirectInput** перед чтением данных с устройства должно "захватить"(**Acquire**) его. При чтении данных обязательно проверьте результат - приложение может "уступить" (**Unacquire**) устройство, например, при потере фокуса приложением (Кстати, это происходит автоматически, если приложение имеет "активный" доступ к устройству). Для захвата и

освобождения устройств существуют функции **Acquire()** и **Unacquire()** соответственно.

А теперь мы готовы использовать то, что написали. Давайте теперь посмотрим на саму процедуру чтения клавиатуры:

```
;#########################################################################
; DI_Read_Keyboard  Procedure
;#########################################################################
DI_Read_Keyboard  PROC

 ;=================================================================
 ; Эта процедура считает клавиатуру и установит входное состояние
 ;=================================================================

 ;============================
 ; Считать, если существует
 ;============================
 .IF lpdikey != NULL
  ;========================
  ; Теперь считаем состояние
  ;========================
  DIDEVINVOKE GetDeviceState, lpdikey, 256, ADDR keyboard_state
  .IF EAX != DI_OK
   JMP err
  .ENDIF
 .ELSE
  ;============================================
  ; клавиатура не включена, обнулить состояние
  ;============================================
  DIINITSTRUCT ADDR keyboard_state, 256
  JMP err

 .ENDIF

done:
 ;===================
 ; Все прошло успешно!
 ;===================
 return TRUE

err:
 ;===================
 ; Упс... ошибка!!! :(
 ;===================
 return FALSE

DI_Read_Keyboard ENDP
;#########################################################################
; END DI_Read_Keyboard
;#########################################################################
```

А здесь мы сначала проверяем, верно ли получен интерфейс устройства. Если нет, то это могло произойти по некоторым причинам, например, отсутствует или не подключена клавиатура, неисправен сам порт, и т.д. Это, вряд ли, случится, но все же лучше убедиться.

Ну а если, интерфейс устройства получен верно, то получаем данные с устройства (в нашем случае с клавиатуры).

По умолчанию клавиатура использует непосредственные (не буферизированные) данные - для того, чтобы прочитать состояние клавиатуры нужно воспользоваться функцией **GetDeviceState()**. Параметрами для этой функции в случае непосредственных данных, будет являться размер массива (равный 256) и указатель на массив из 256 байт для данных о всех клавишах. Для удобства работы, для каждого индекса массива, в файле **"DInput.inc"** определены константы с именами клавиш, например: **DIK_J, DIK_N, DIK_ESCAPE,** и т.д.
Для определения нажата ли клавиша, достаточно проверить соответствующий ей байт, и если он имеет значение **TRUE** (или не равен 0), то значит клавиша нажата:

```
 .if keyboard_state[DIK_ESCAPE]
     ; клавиша ESC нажата
 .else
     ; клавиша ESC не нажата
 .endif
```

От переводчика. Обратите внимание, как автор проверяет, нажата ли клавиша. Если значение не равно 0, то считается, что клавиша нажата.

Так, конечно, работать будет, но в официальной документации по **DirectX** всё же говорится, что мы должны проверять только старший (последний) бит байта, и только если он установлен, то считать, что клавиша нажата.

Таким образом, правильнее было бы написать вот так:

```
.if keyboard_state[DIK_ESCAPE] & 80h
    ; клавиша ESC нажата
.else
    ; клавиша ESC не нажата
.endif
```

Ну ладно, не будем задерживаться на мелочах и пойдем дальше.
На очереди у нас процедура завершения **DirectInput**:

```
;#########################################################################
; DI_ShutDown Procedure
;#########################################################################
DI_ShutDown PROC

 ;=======================================================
 ; Эта процедура завершает DirectInput
 ;=======================================================

 ;============================
 ; Освобождаем мышь
 ;============================
 DIDEVINVOKE Unacquire, lpdimouse
 DIDEVINVOKE Release, lpdimouse

 ;============================
 ; Освобождаем клавиатуру
 ;============================
 DIDEVINVOKE Unacquire, lpdikey
 DIDEVINVOKE Release, lpdikey

 ;================================
 ; Удаляем объект DirectInput
 ;================================
 DIINVOKE Release, lpdi

done:
 ;==================
 ; Успешно завершено! :)
 ;==================
 return TRUE

err:
 ;==================
 ; Упс!... Ошибка! :(
 ;==================
 return FALSE

DI_ShutDown ENDP
;#########################################################################
; END DI_ShutDown
;#########################################################################
```

Что же собственно делает эта процедура?

По завершению приложения мы должны **"убрать за собой" - "уступить"**(освободить) все устройства (которые захватили при инициализации), и удалить объекты **DirectInput** вызовом **Release()**.

В действительности, **DirectInput** - одна из самых простых частей **DirectX**. И мы разобрались, как с ней работать. Теперь нам не обязательно зависеть от "оконной" процедуры и сообщений **Windows** - мы можем в любом месте программы и в любой промежуток времени прочитать данные с устройства.

Вот и все, теперь у нас есть все, что нам нужно для создания системы меню. Но прежде, чем мы этим займемся, давайте разберемся с синхронизацией.

-- > Синхронизация и Виндоуз

Тот, кто только что написал или начал писать свою первую игру обязательно обнаружит, что на компьютерах с разной конфигурацией или частотой процессора его игра будет работать по-разному. Кто-то сгоряча начнет добавлять циклы задержки. Ха! А теперь представьте, как это будет выглядеть на более медленных машинах? Кто-то будет определять или запрашивать частоту процессора... Все это неверно, ведь производительность зависит не только от ЦП. А как же тогда? Ответ прост, да и наверное вы его уже знаете - отталкиваться от пройденного времени.

Для определения времени в Windows можно воспользоваться следующими вызовами:

• **GetTickCount()** Функция **kernel32.dll** (ядро) возвращает количество миллисекунд прошедших с момента старта Windows. Возвращаемое значение переполняется приблизительно через 49,71 дней и отсчет начинается снова (это надо учитывать!).

• **timeGetTime** Функция **winmm.dll**(мультимедия-**API**) является копией функции **GetTickCount**. Возможно регулирование точности возвращаемого значения - снижение точности более 1 миллисекунды. **[примечание]** по утверждению автора эта функция более точная и надежная, чем **GetTickCount**

• Ну а если нужна большая точность, то можно воспользоваться вызовами **QueryPerformanceFrequency** и **QueryPerformanceCounter**, или как их еще называют, счетчиком производительности. Но, к сожалению, он поддерживается не всеми системами, а точнее процессорами. По сути это 64-х битный счетчик тактовых импульсов процессора (по моему он называется **TSC**), который есть в последних **x86**-совместимых процессорах (в **Celeron** и **Pentium** точно есть). По Микрософтовской документации функция **QueryPerformanceCounter** считывает в предоставленную Вами **64-битную переменную** значение **TSC**, если он есть, либо 0, если его нет. А если быть немножко поточнее, то функция **QueryPerformanceCounter** возвращает не количество тактов процессора, а количество тиков, каждый из которых равен примерно 0,838 мкс. Хотя перед тем, как пользоваться этой функцией, нужно с помощью **QueryPerformanceFrequency** узнать частоту инкремента этого счетчика для данного компьютера (также 64-х битное значение). **[примечание]** Автор этой статьи, называет его высокоэффективным таймером (**High Performance**), отсюда и в названии переменных, связанных с этим таймером, присутствует аббревиатура HP.

Итак, для начала нам потребуется процедура инициализации нашей системы синхронизации - **Init_Time()** :

```
;#########################################################################
; Init_Time Procedure
;#########################################################################
Init_Time PROC

 ;=====================================================
 ; Эта функция определяет,
 ; можем ли мы использовать счетчик производительности.
 ; и устанавливает необходимые переменные.
 ;=====================================================
 ;=========================================
```

```
 ; извлекаем  частоту счетчика производительности,
 ; если таковой существует.
 ;=========================================
 INVOKE QueryPerformanceFrequency, ADDR HPTimerVar

 .IF EAX == FALSE
  ;====================
  ; не использовать
  ;====================
  MOV UseHP, FALSE
  JMP done

 .ENDIF

 ;=======================================
 ; Мы можем его использовать,
 ; так что устанавливаем переменные и частоту.
 ;=======================================
 MOV UseHP, TRUE
 MOV EAX, HPTimerVar
 MOV HPTimerFreq, EAX
 MOV ECX, 1000
 XOR EDX, EDX
 DIV ECX
 MOV HPTicksPerMS, EAX

done:
 ;===================
 ; Завершаем процедуру
 ;===================
 return TRUE

Init_Time ENDP
;########################################################################
; END Init_Time
;########################################################################
```

Эта процедура определяет, поддерживает ли установленное аппаратное обеспечение счетчик производительности, и в случае успеха, возвращает его частоту. Для чего и используется функция **QueryPerformanceFrequency**.
Вот ее описание:

BOOL QueryPerformanceFrequency (LARGE_INTEGER *lpFrequency);

• **lpFrequency** - указывает на 64-х битную переменную, значение которой, в отсчетах в секунду, функция устанавливает в текущую частоту счетчика производительности. Если установленное аппаратное обеспечение не поддерживает счетчик производительности, значение этого параметра может быть равно нулю.
• **Возвращаемое значение:** В случае, если установленное аппаратное обеспечение поддерживает счетчик производительности, возвращается ненулевое значение (**TRUE**). В случае, если установленное аппаратное обеспечение не поддерживает счетчик производительности, возвращается нуль (**FALSE**).

Теперь предположим, что счетчик производительности у нас есть, тогда код сохраняет его частоту и подсчитывает количество тиков в миллисекунду.

Следующая функция - **Start_Time()**. Эта функция используется, для запуска таймера с переданным ей значением. Она будет использоваться, для управления скоростью смены кадров.
А вот собственно и сам код...

```
;########################################################################
; Start_Time Procedure
;########################################################################
Start_Time PROC ptr_time_var:DWORD

 ;=====================================================
```

```
 ; Эта функция запускает таймер и сохраняет значение,
 ; по адресу переданному ей в качестве параметра.
 ;=====================================================

 ;=======================================
 ; Доступен ли нам счетчик производительности?
 ;=======================================
 .IF UseHP == TRUE
  ;==================================
  ; Да, доступен.
  ;==================================
  INVOKE QueryPerformanceCounter, ADDR HPTimerVar
  MOV EAX, HPTimerVar
  MOV EBX, ptr_time_var
  MOV DWORD PTR [EBX], EAX

 .ELSE
  ;==================================
  ; Нет. Вместо него используем  timeGetTime.
  ;==================================

  ;==================================
  ; Получаем начальное время.
  ;==================================
  INVOKE timeGetTime

  ;==================================
  ; Устанавливаем переменную
  ;==================================
  MOV EBX, ptr_time_var
  MOV DWORD PTR [EBX], EAX

 .ENDIF

done:
 ;===================
 ; завершаем
 ;===================
 return TRUE

Start_Time ENDP
;#########################################################################
; END Start_Time
;#########################################################################
```

Этот код очень прост. Если счетчик производительности у нас доступен, то вызываем **QueryPerformanceCounter()**, а если недоступен, то вместо него вызываем **timeGetTime()**.

Что же делают эти функции?
Функция **QueryPerformanceCounter()**, вот ее описание: BOOL QueryPerformanceCounter (LARGE_INTEGER *lpPerformanceCount);*

• **lpPerformanceCount** - указывает на 64-х битную переменную, которую функция устанавливает в текущее значение счетчика. Если установленное аппаратное обеспечение не поддерживает счетчик производительности, этот параметр может быть установлен в нуль.

Функция **timeGetTime()**, вот ее описание: **DWORD timeGetTime (VOID);**

• Функция возвращает текущее системное время в миллисекундах.

И еще одна, последняя процедура **Wait_Time()**. Это однотипная процедура для **Start_Time()**. И вместе они используются для управления скоростью смены кадров, нашей игры.
А вот и сам код:

```
;#########################################################################
; Wait_Time Procedure
;#########################################################################
Wait_Time PROC time_var:DWORD, time:DWORD
 ;=====================================================
```

```
; Эта функция ожидает прохождения определенного (в переменной time)
;  промежутка времени от начального времени (time_var).
; Функция возвращает время (в миллисекундах),
; затраченное на ее выполнение.
;=======================================================

;=======================================
; Используем ли мы счетчик производительности
;=======================================
.IF UseHP == TRUE
 ;==================================
 ; Да, используем
 ;==================================

 ;==================================
 ; Корректируем время для частоты
 ;==================================
 MOV EAX, 1000
 MOV ECX, time
 XOR EDX, EDX
 DIV ECX
 MOV ECX, EAX
 MOV EAX, HPTimerFreq
 XOR EDX, EDX
 DIV ECX
 MOV time, EAX

 ;================================
 PUSH EAX

again1:

 ;================================
 POP EAX

 ;====================================
 ; Получаем текущее время
 ;====================================
 INVOKE QueryPerformanceCounter, ADDR HPTimerVar
 MOV EAX, HPTimerVar

 ;====================================
 ; Вычитаем из него начальное время
 ;====================================
 MOV ECX, time_var
 MOV EBX, time
 SUB EAX, ECX

 ;====================================
 ; Сохраняем требуемое количество времени
 ;====================================
 PUSH EAX

 ;====================================
 ; Прыгаем наверх и выполняем цикл снова,
 ; если значение в eax меньше или равно,
 ; чем значение в переменной time
 ;====================================
 SUB EAX, EBX
 JLE again1

 ;=======================================
 ; Востанавливаем конечное время из стека
 ;=======================================
 POP EAX

 ;=======================================
 ; Переводим в миллисекунды (MS)
 ;=======================================
 MOV ECX, HPTicksPerMS
 XOR EDX, EDX
```

```
    DIV ECX

   .ELSE
    ;=================================
    ; Нет. Счетчик производительности недоступен
    ; вместо него используем timeGetTime.
    ;=================================

    ;=================================
    PUSH EAX

   again:
    ;=====================================
    POP EAX

    ;=====================================
    ; Получаем текущее время
    ;=====================================
    INVOKE timeGetTime

    ;=====================================
    ; Вычитаем из него начальное время
    ;=====================================
    MOV ECX, time_var
    MOV EBX, time
    SUB EAX, ECX

    ;=====================================
    ; Сохраняем, требуемое количество времени
    ;=====================================
    PUSH EAX

    ;=====================================
    ; Прыгаем наверх и выполняем цикл снова,
    ; если значение в eax меньше или равно,
    ; чем значение в переменной time
    ;=====================================
    SUB EAX, EBX
    JLE again

    ;=======================================
    ; Вытаскиваем конечное время из стека
    ;=======================================
    POP EAX

   .ENDIF

   ;=======================
   ; Возвращаемся
   ;=======================
   RET

  Wait_Time ENDP
  ;#########################################################################
  ; END Wait_Time
  ;#########################################################################
```

Эта процедура возможно самая сложная из описанных ранее. Что же она делает? Она ожидает прохождения определенного промежутка времени (переданного вторым параметром), от начального времени (переданного первым параметром). Другими словами, например, начальное время у нас было 100мс, и мы указали ожидать 50мс, то процедура не вернет управление, пока текущее время не станет больше или равно 150мс. Процедура возвращает время, затраченное на ее выполнение.

Вот и все о синхронизации, теперь мы можем перейти к нашему меню.

-- > Меню

Создание меню в игре, может быть довольно сложным занятием. Но, в нашем случае это будет легко. Основная идея состоит в том, чтобы обеспечить возможность выбора процесса в игре. В большинстве случаев, меню состоит из фонового битмапа и битмапа - "указателя", для выбора пункта меню. Я все же решил сделать это немного по-другому. Вместо "указателя", мы просто назначим соответствующие клавиши каждому пункту меню.

Есть еще одна важная вещь, это выбор типа системы меню. Вы хотите, чтобы код вошел в "цикл меню" и не возвращал управление, пока пользователь не сделает выбор? Или, Вы хотите вызывать функцию меню снова и снова? В случае с Windows, второй тип в миллион раз лучше, так как нам нужно обрабатывать сообщения. Если мы составим программу первым способом, то представьте, что произойдет, если пользователь нажмет "**ALT+TAB**". Возможно программа повиснет. (ПРИМЕЧАНИЕ: в игре пока нет поддержки "**ALT+TAB** ".... но это будет реализовано в более поздних выпусках!). Так что, будем использовать второй тип системы.

В процедурах инициализации и завершения, нет ничего, чего бы вы не видели ранее. Все, что они делают, это загружают два фоновых битмапа для нашего меню. Один из них для главного меню, а второй для файлового меню. А процедура завершения просто освобождает выделенную под них память.

Интересный код находится в процедурах **Process_XXXX_Menu()**. Давайте подробно рассмотрим одну из них, процедуру **Process_Main_Menu()**. Ну, и как обычно, вот ее код:

```
;#########################################################################
; Process_Main_Menu Procedure
;#########################################################################
Process_Main_Menu PROC

 ;=============================================================
 ; Эта процедура обрабатывает главное меню нашей игры
 ;=============================================================

 ;===================================
 ; Блокируем DirectDraw back buffer
 ;===================================
 INVOKE DD_Lock_Surface, lpddsback, ADDR lPitch

 ;============================
 ; Проверяем на ошибки.
 ;============================
 .IF EAX == FALSE
  JMP err
 .ENDIF

 ;===================================
 ; Рисуем битмап на поверхность
 ;===================================
 INVOKE Draw_Bitmap, EAX, ptr_MAIN_MENU, lPitch, screen_bpp

 ;===================================
 ; Разблокируем back buffer
 ;===================================
 INVOKE DD_Unlock_Surface, lpddsback

 ;============================
 ; Проверяем на ошибки
 ;============================
 .IF EAX == FALSE
  JMP err
 .ENDIF

 ;====================================
 ; Все OK. Так что переключаем поверхности
 ; и делаем видимой поверхность, на которой,
 ; только что, нарисовали битмап.
 ;====================================
 INVOKE DD_Flip
```

```
 ;============================
 ; Проверяем на ошибки
 ;============================
 .IF EAX == FALSE
  JMP err
 .ENDIF

 ;=========================================================
 ; Теперь читаем клавиатуру и проверяем нажаты ли
 ; клавиши, соответствующие нашему меню.
 ;=========================================================
 INVOKE DI_Read_Keyboard

 .IF keyboard_state[DIK_N]
  ;======================
  ; Новая игра.
  ;======================
  return MENU_NEW

 .ELSEIF keyboard_state[DIK_G]
  ;======================
  ; Файлы игры.
  ;======================
  return MENU_FILES

 .ELSEIF keyboard_state[DIK_R]
  ;======================
  ; Возврат.
  ;======================
  return MENU_GAME

 .ELSEIF keyboard_state[DIK_E]
  ;======================
  ; Выход.
  ;======================
  return MENU_EXIT

 .ENDIF

done:
 ;===================
 ; Выходим ничего не делая
 ;===================
 return MENU_NOTHING

err:
 ;===================
 ; В процессе выполнения
 ; возникли ошибки. :(
 ;===================
 return MENU_ERROR

Process_Main_Menu ENDP
;#########################################################################
; END Process_Main_Menu
;#########################################################################
```

Интересная процедура, не так ли? Ну хорошо... возможно и нет. И что же она делает?

Начинается она с блокировки фоновой поверхности (**back buffer**), и прорисовки на ней битмапа меню. Затем разблокируем фоновую поверхность, и переключаем поверхности, для того, чтобы мы могли ее увидеть.

Вот мы и добрались до вызова одной из наших **DirectInput** процедур, до вызова процедуры **DI_Read_Keyboard()**. Эта процедура, как вы помните, получает состояние всех клавиш на клавиатуре. После ее вызова, мы проверяем, были ли нажаты интересующие нас клавиши. Если да, то возвращаем значение соответствующее нажатой клавише. Например, если пользователь нажимает клавишу '**N**' для новой игры, то мы возвращаем значение **MENU_NEW** вызывающей

программе. Эти значения известны и определены в начале модуля.

А если же ничего нажато не было, то мы просто возвращаем значение **MENU_NOTHING**. А если при выполнении кода произошли ошибки, то возвращаем значение **MENU_ERROR**.

Тот же метод используется и для процедуры **Process_File_Menu()**. Вот мы и связали код **DirectInput** с нашей системой меню. Все, что нам осталось сделать, это связать меню с кодом таймера. **-- > Связываем все вместе**

Вот мы почти и закончили. Пришло время связать все, что мы написали и поместить в один пакет. Так что давайте начнем с процедуры инициализации игры.

```
;#########################################################################
; Game_Init Procedure
;#########################################################################
Game_Init PROC

 ;=======================================================
 ; Процедура инициализации игры
 ;=======================================================

 ;===========================================
 ; Инициализация Direct Draw -- 640, 480, bpp
 ;===========================================
 INVOKE DD_Init, 640, 480, screen_bpp

 ;============================
 ; Проверяем на ошибки
 ;============================
 .IF EAX == FALSE
  JMP err
 .ENDIF

 ;===========================================
 ; Читаем битмап и создаем буфер
 ;===========================================
 INVOKE Create_From_SFP, ADDR ptr_BMP_LOAD, ADDR szLoading, screen_bpp

 ;============================
 ; Проверяем на ошибки
 ;============================
 .IF EAX == FALSE
  JMP err
 .ENDIF

 ;===========================================
 ; Блокируем фоновый буфер DirectDraw
 ;===========================================
 INVOKE DD_Lock_Surface, lpddsback, ADDR lPitch

 ;============================
 ; Проверяем на ошибки
 ;============================
 .IF EAX == FALSE
  JMP err
 .ENDIF

 ;===========================================
 ; Прорисовываем битмап на поверхности
 ;===========================================
 INVOKE Draw_Bitmap, EAX, ptr_BMP_LOAD, lPitch, screen_bpp

 ;===========================================
 ; Разблокируем фоновый буфер
 ;===========================================
 INVOKE DD_Unlock_Surface, lpddsback

 ;============================
 ; Проверяем на ошибки
 ;============================
```

```
 .IF EAX == FALSE
  JMP err
 .ENDIF

 ;=============================================
 ; Все OK! Так что переключаем поверхности
 ; и делаем видимой поверхность, на которой,
 ; только что, нарисовали битмап.
 ;=============================================
 INVOKE DD_Flip

 ;============================
 ; Проверка на ошибки
 ;============================
 .IF EAX == FALSE
  JMP err
 .ENDIF

 ;=============================================
 ; Инициализация Direct Input
 ;=============================================
 INVOKE DI_Init

 ;============================
 ; Проверка на ошибки
 ;============================
 .IF EAX == FALSE
  JMP err
 .ENDIF

 ;=============================================
 ; Инициализация системы синхронизации
 ;=============================================
 INVOKE Init_Time

 ;=============================================
 ; Инициализация наших меню
 ;=============================================
 INVOKE Init_Menu

 ;============================
 ; Проверка на ошибки
 ;============================
 .IF EAX == FALSE
  JMP err
 .ENDIF

 ;=============================================
 ; Устанавливаем режим меню
 ;=============================================
 MOV GameState, GS_MENU

 ;=============================================
 ; Освобождаем память битмапа
 ;=============================================
 INVOKE GlobalFree, ptr_BMP_LOAD

done:
 ;============================
 ; Успешное завершение
 ;============================
 return TRUE

err:
 ;============================
 ; В процессе выполнения
 ; возникли ошибки :(
 ;============================
 return FALSE

Game_Init ENDP
```

```
;#########################################################################
; END Game_Init
;#########################################################################
```

Эта процедура претерпела некоторые изменения, с тех пор, как вы ее последний раз видели. Сначала мы добавили несколько вызовов: один для инициализации нашей системы синхронизации, один для инициализации нашей системы меню, и еще один для инициализации нашей **DirectInput** библиотеки. А также, в конце процедуры, мы освобождаем память, выделенную под битмап загрузочного экрана. Это сделано для того, чтобы не использовать зря память, под битмап, который нам больше не понадобится. И еще одна вещь, на которую стоит обратить внимание, это то, что мы добавили глобальную переменную **GameState**, которая хранит текущее состояние игры, а также указывает главному игровому циклу, какое состояние обрабатывать. В конце процедуры инициализации игры, мы устанавливаем эту переменную в значение **GS_MENU**.

Вот собственно и все, что было изменено в процедуре инициализации. А в код завершения были добавлены вызовы процедур завершения для модуля **DirectInput** , и для модуля меню. А еще, мы должны сделать изменения в процедуре главного игрового цикла. Фактически, нам достаточно просто заменить старую процедуру новой, так как, с прошлых уроков она была пустой (помните, мы ее просто зарезервировали, но не использовали?).

А вот и сам новый код процедуры главного игрового цикла:

```
;#########################################################################
; Game_Main Procedure
;#########################################################################
Game_Main PROC

 ;=============================================================
 ; Это процедура - сердце игры (основа игры),
 ; она получает управление снова и снова,
 ; даже если мы обрабатываем сообщение!
 ;=============================================================

 ;=======================================
 ; Локальные переменные
 ;=======================================
 LOCAL StartTime :DWORD

 ;====================================
 ; Получает стартовое время для цикла
 ;====================================
 INVOKE Start_Time, ADDR StartTime

 ;=================================================================
 ; Выбирает нужное действие(я), исходя из значения переменной GameState
 ;=================================================================
 .IF GameState == GS_MENU
  ;================================
  ; Мы находимся в режиме главного меню
  ;================================
  INVOKE Process_Main_Menu

  ;================================
  ; Что хочет сделать пользователь
  ;================================
  .IF EAX == MENU_NOTHING
   ;================================
   ; пользователь ничего не выбирал, так что,
   ; соответственно, ничего и не делаем
   ;================================

  .ELSEIF EAX == MENU_ERROR
   ;================================
   ; А тут мы получили код ошибки
   ;================================

  .ELSEIF EAX == MENU_NEW
```

```
  ;==================================
  ; Пользователь решил начать новую игру
  ;==================================

 .ELSEIF EAX == MENU_FILES
  ;==================================
  ; Пользователю понадобилось файловое меню
  ;==================================
  MOV GameState, GS_FILE

 .ELSEIF EAX == MENU_GAME
  ;==================================
  ; Пользователь хочет вернуться
  ;==================================

 .ELSEIF EAX == MENU_EXIT
  ;==================================
  ; пользователь надумал покинуть игру
  ;==================================
  MOV GameState, GS_EXIT

 .ENDIF


.ELSEIF GameState == GS_FILE
 ;==================================
 ; Мы находимся в состоянии файлового меню
 ;==================================
 INVOKE Process_File_Menu

 ;==================================
 ; И чего же хочет пользователь?
 ;==================================
 .IF EAX == MENU_NOTHING
  ;==================================
  ; Пользователь ничего пока не выбрал (думает! :)
  ; так что, ничего не делаем.
  ;==================================

 .ELSEIF EAX == MENU_ERROR
  ;==================================
  ; А тут мы окажемся в случае ошибки
  ;==================================

 .ELSEIF EAX == MENU_LOAD
  ;==================================
  ; Пользователь решил загрузить игру
  ;==================================

 .ELSEIF EAX == MENU_SAVE
  ;==================================
  ; Пользователь захотел сохранить игру
  ;==================================


 .ELSEIF EAX == MENU_MAIN
  ;==================================
  ; Пользователь хочет вернуться
  ;==================================
  MOV GameState, GS_MENU

 .ENDIF


.ELSEIF GameState == GS_PLAY
 ;==================================
 ; А здесь мы находимся в режиме игры
 ;==================================

.ELSEIF GameState == GS_DIE
 ;==================================
```

```
    ; мы проиграли :(
    ;================================

   .ENDIF

   ;==================================
   ; Ожидаем синхронизации времени
   ;==================================
   INVOKE Wait_Time, StartTime, sync_time

  done:
   ;===================
   ; Выполнено без ошибок!
   ;===================
   return TRUE

  err:
   ;===================
   ; В процессе выполнения
   ; возникли ошибки :(
   ;===================
   return FALSE

  Game_Main ENDP
  ;########################################################################
  ; END Game_Main
  ;########################################################################
```

Первое, на что вы должны обратить внимание, это то, что код расположен между вызовами процедур **Start_Time()** - наверху, и **Wait_Time()** - внизу. Эти процедуры управляют скоростью кадров в нашей игре. Я сделал так, чтобы было **25 FPS** (т.е. 25 кадров в секунду), или 40 миллисекунд.

Далее мы имеем одну большую конструкцию **IF-ELSE**, которая выбирает нужные действия, исходя из значения глобальной переменной **GameState** , которую мы специально и определили для этой цели.

Новый игровой цикл, это просто менеджер состояний. Он выполняет соответствующий текущему состоянию код. Все игры имеют нечто похожее в их ядре.

-- > До следующего раза ... Ура! Вот у нас и готов хороший, чистый каркас для игры, в котором фактически не хватает самого игрового кода. Но, для этого вам придется ждать следующей статьи.
-- > В следующей статье...

В следующей статье мы охватим еще некоторый расширенный материал. Также сделаем упор непосредственно на сам игровой код, включая: основу анимации, "цикл", и, ... как обычно, ... что-нибудь еще, что придет в голову.

Другая вещь, о которой я должен упомянуть, состоит в том, что эта игра является неполной. Я знаю, что это очевидно, но многие из Вас вероятно задаются вопросом, почему нет никаких переходов, звуков, или "крутых спецэффектов" в игре. Ответ ..., потому что я и не стремился к этому. Но, если честно, то я планирую охватить это все. Ну, а для тех из Вас, кто более продвинут, и думает, что я иду слишком медленно, потерпите. Хороший материал впереди... Я обещаю.

--> Готовый исходник для этой статьи, можно взять здесь...

А для его компиляции вам также понадобятся **LIB'ы DirectX**, которые можно найти в **DirectX SDK**, либо скачать здесь... или здесь....[от переводчика]
P.S. **Вот и все, наконец-то я выкроил время и перевел 3-ю часть (а всего их 6 !!!) данного туториала, и в будущем, планирую перевести остальные 3. Я также, хотел бы поблагодарить всех, кто откликнулся на прошлые статьи.** P.S.S. В интернете очень много статей о том, как писать вирусы, и почти совсем нет информации о том, как создавать красивые игры, демо-сцены (демки), заставки и т.д., как реализовать некоторые алгоритмы, например: вода (Java апплет - Lake

помните такой? А как такое же сделать на ассемблере под windows? Никто не задумывался?), как реализовать jpeg/mpeg/mp3 декодер, как реализовать огонь и многое другое (имеется в виду под windows и на чистом ассемблере)? Нет, есть конечно некоторые исходники реализации плазмы, огня и т.д., но ведь в них почти нет никаких комментариев, и начинающему демо-мэйкеру разобраться в огромных листингах кода очень тяжело.

Так почему же такой информации очень и очень мало, по сравнению с информацией о создании вирусов??? Да потому, что вирусы гораздо легче писать, правда я не понимаю для чего? Неужели вам станет легче, если кто-то пострадает от вашего вируса??? У всех вирусов почти один и тот же алгоритм: открыть файл, дописать в него тело вируса, закрыть файл, ну и возможно выполнить какие-нибудь деструктивные действия (типа убить инфу на винте), или стырить чей-нибудь пароль к интернету и т.д. А вот чтобы реализовать какой-нибудь видеоэффект, то тут надо сильно постараться, возможно даже подучить (или вспомнить) математику, геометрию, физику... к тому же, это намного интереснее! Это надо как-то исправлять!!! Надо учить начинающих ассемблерщиков не созданию вирусов, а созданию чего-нибудь более стоящего. Ох, что-то меня на философию потянуло... надо завязывать. И к чему я все это рассказываю? Ах да, если у вас есть свои разработки, свои программы, свои игры, которыми вы могли бы поделиться с людьми, то присылайте их мне (мой e-mail ниже), желательно с подробным (или хотя бы с кратким) описанием.

А также по всем вопросам (*для вопросов относящихся к программированию есть форум!*), предложениям, пожеланиям и т.д. пишите:

e-mail: **unis2@mail.ru**, **Andrey aka UniSoft** Вот теперь все!!! До скорых встреч!!! **Счастливого кодирования!!!***Особая благодарность CyberManiac за помощь в оформлении статьи и за исправление ошибок!!!*

DirectX 8.1 в MASM32: Урок 0

Автор: keYMax

Дата: 2003-02-26

Предисловие.

Привет всем ! Сидел я как-то в инете и наткнулся на сайт www.vvsu.ru/dkscs, а там, на цикл статей Дениса "Mr.Snow" Кожухова (в данный момент сайт переехал на Xdev.ru). Статьи были посвящены программированию Direct3D 8.1 на языке C++. Посмотрел я на них и подумал, а что если забацать тоже самое, но на MASM. Решение пришло из-за того, что меня несколько удивляет факт наличия в инете множества исходников на асме, использующих OpenGL, и совсем мало использующих Direct3D, да и то старых версий (может я не там искал ?). Конечно, OpenGL это рулез, скажете вы, а Microsoft с его DirectX полный отстой и мастдай форева. Таких "товарисчей" я попрошу сразу удалиться даже не читая дальше. На мой взгляд, DirectX не такой уж плохой API и на нем тоже можно что-нибудь состряпать, поэтому я постараюсь в меру своих сил и возможностей опубликовать цикл уроков по Direct3D 8.1, на этом сайте (если организаторы позволят ;)).

В общем, всем кому интересно прошу далее....

Урок 0. Создаем каркас приложения WIN32 для будущих экспериментов !

Чтобы было с чего начинать, я написал каркас приложения, который буду использовать во всех следующих уроках. Из каркаса я выбросил все лишнее, чтобы все внимание сосредоточить на работе с DIRECTX и не отвлекаться на всякие WinMain и прочую ерунду.

Первое, что я сделал, это вынес структуру класса в секцию DATA и сразу заполнил то, что можно. Меньше кода и более понятно я думаю.

Во-вторых, я убрал функцию WinMain. А также написал цикл обработки сообщений командами ассемблера, а не псевдокомандами .IF .ELSE и т.д.

В третьих оставил только два события: CREATE и DESTROY.

В итоге совсем немного текста !

```
Для постройки приложения был использован пакет MASM32.
Версия ML 6.15.8803
Версия LINK 5.12.8078

Все исходники редактировались в ASM Editor'e.
```

Исходник прилагается (находится в архиве `wasm_files.zip`: `files/0230/dx8masm00.zip`).

Всем пока до следующего урока :)

В следующем уроке мы создадим объект DIRECT3D и осуществим небольшой опыт, чтобы проверить его работоспособность.

Авторство принадлежит Пономареву Михаилу ака keYMax.

Все вопросы и ругательства слать по адресу mybox@aib.ru

DirectX 8.1 в MASM32: Урок 1

Автор: keYMax

Дата: 2003-03-01

Самое главное, что нам потребуется для работы с DirectX это заголовочные файлы и библиотеки. Можно взять их из Microsoft DirectX SDK 8.1. Все, что нам оттуда нужно, это только папка LIB. У меня не было возможности взять SDK с компакта, а качать более 100 метров ради пары десятков файлов полный идиотизм. Хоть мелгомягкие и запрещают выкладывать какие либо части их SDK, находятся еще добрые люди на свете и набрав в поисковике искомое мы всегда получим парочку адресов. Все ленивые могут посмотреть по адресу упомянутому мною в предыдущем уроке. До последнего момента там было то что нам нужно.

Так как SDK рассчитан на работу с языком C++, то заголовочные файлы необходимо адаптировать. Не переживайте. Для данных уроков я уже адаптировал все необходимые файлы и надеюсь, работа будет продолжена. Адаптированные INC файлы включены в исходник уроков, забирайте и пользуйтесь. Формат библиотек является родным и для C++ и для MASM, поэтому библиотеки подходят без проблем. Ну вот и все приготовления и прежде, чем мы перейдем собственно к нашему уроку, хотелось бы еще упомянуть, что все относящееся к DirectX я буду сопровождать подробной справкой. Справка является моим корявым переводом справочника из SDK, поэтому не ругайте меня за возможные ошибки.

Все что у нас сейчас имеется это приложение, при запуске выводящее на экран окошко размером 320x240, и нам надо заставить работать в этом окне Direct3D. Как это сделать? Нет ничего проще!

Первое что мы добавим это подключение необходимых файлов:

```
include   d3d8.inc
include   d3d8caps.inc
include   d3d8types.inc
includelib   d3d8.lib
```

Теперь обьявим необходимые переменные и структуры:

```
Clearcolor  DWORD   0 ; Переменная для очистки экрана цветом
Zvalue  REAL4    1.0 ; Значение для Z буфера ( см.ниже )

pd3d  DWORD    ? ; Указатель на Direct3D
pd3dDevice DWORD   ? ; Указатель на Direct3DDevice
d3ddm  D3DDISPLAYMODE       <> ; Структура для параметров экрана
d3dpp   D3DPRESENT_PARAMETERS <> ; Структура для параметров нашего Direct3Ddevice
```

Справка

```
D3DDISPLAYMODE STRUC

   Width1  UINT             ?   ; Ширина экрана в пикселях
   Height  UINT             ?   ; Высота экрана в пикселях
   RefreshRate   UINT  ?   ; Частота обновления. 0 означает частота по умолчанию
   Format  DWORD  ?   ; Формат использ. поверхности ( см. D3d8Types.inc )

D3DDISPLAYMODE ENDS

D3DPRESENT_PARAMETERS STRUC

   BackBufferWidth  UINT  ? ; Ширина BackBuffer. 0 если работаем в окне.
   BackBufferHeight  UINT  ? ; Высота BackBuffer. 0 если работаем в окне.
   BackBufferFormat  DWORD  ? ; Формат использ. поверхности ( см. D3d8Types.inc )
   BackBufferCount  UINT  ? ; 0,1,2 или 3 - число буферов. 0 трактуется как 1.
       ; Если указанное число буферов не может быть
       ; создано то помещается число буферов которые
       ; были реально созданы
   MultiSampleType  DWORD  ?  ; 2-16 уровни мультисэмплинга картинки.
```

```
        ; 1 не используется, а 0 равносилен отсутствию
        ; мультисэмплинга ( D3DMULTISAMPLE_NONE )
        ; Мультисэмплинг возможен только если установлен
        ; D3DSWAPEFFECT_DISCARD. В противном случае
        ; здесь должен быть 0.
    SwapEffect   DWORD  ?  ; Эффект обмена поверхностей.
        ; Если мы работаем в окне и используем
        ; D3DSWAPEFFECT_FLIP, тогда будет создан один
        ; дополнительный BackBuffer. Пока один буфер
        ; показывается на экране можно рисовать в другой.
        ; Потом они меняются местами и т.д
        ; D3DSWAPEFFECT_COPY и
        ; D3DSWAPEFFECT_COPY_VSYNC требуют чтобы
        ; BackBufferCount был установлен в 1. ( содержимое
                                                                ; буфера копируется на экран без
    ожидания
                                                                ; обратного хода луча и с
    ожиданием соответственно)
                                                                ; D3DSWAPEFFECT_DISCARD сначала
    буфер
                                                                ; заполняется а потом
    показывается на экран
    hDeviceWindow  HWND         ?  ; ID нашего окошка
    Windowed   BOOL            ?  ; Режим работы. 0 - полноэкранный, 1 - в окошке
    EnableAutoDepthStencil BOOL            ?  ; Если значение установить в 1, то Direct3D будет
    управлять
        ; для приложения буфером глубины. Будет создан Z-Stencil
        ; буфер. При этом в AutoDepthStencilFormat должен быть
        ; установлен правильный формат поверхности.
                                                                ; Если 0 - ничего не будет
    создано
    AutoDepthStencilFormat DWORD  ?  ; Формат использ. поверхности ( см. D3d8Types.inc )
         ; Игнорир. если EnableAutoDepthStencil не равен 1
    Flags    DWORD         ?  ; 0 или D3DPRESENTFLAG_LOCKABLE_BACKBUFFER=1
         ; Нужен если мы будем блокировать BackBuffer
    FullScreen_RefreshRateInHz UINT  ?  ; Частота обновления экрана в Hz.
                                                                ; Должно быть 0 если мы работаем
    в оконном режиме
          ; Если указать D3DPRESENT_RATE_DEFAULT, то
                                                                ; берется текущее значение, если
    в окошке, и по
                                                                ; умолчанию ( обычно 60 Hz ),
    если в полноэкранном
    FullScreen_PresentationInterval UINT   ?  ; Интервал показа на экране backbuffera
                                                                ; Должно быть 0 если мы работаем
    в оконном режиме
        ; D3DPRESENT_INTERVAL_IMMEDIATE -  рисуем немедленно
        ; не ждя обратного хода луча
                    ; 1 - рисуем ждя обратный ход луча ( FPS - не будет
                                                                ; превышать Hz экрана )
                    ; 2 - FPS равен половине Hz
                    ; 3 - FPS равен трети Hz
                    ; 4 - FPS равен четверти Hz
D3DPRESENT_PARAMETERS ENDS
```

Далее создадим три процедуры Init_Direct3D, Destroy_Direct3D и Render_Scene

итак Init_Direct3D

Создаем обьект Direct3D8:

```
    invoke   Direct3DCreate8, D3D_SDK_VERSION
    mov pd3d, eax
```

Справка

```
Direct3DCreate8 передается в качестве параметра версия SDK.

Метод возвращает указатель на обьект если все прошло успешно.
Если ошибка, то в EAX возвращается код ошибки, какой именно я не знаю :( ( возможно 0 )
```

Затем мы получаем информацию о текущих параметрах экрана:

```
d3d8  GetAdapterDisplayMode, pd3d, D3DADAPTER_DEFAULT, ADDR d3ddm
```

Справка

```
1. Номер видеоадаптера. D3DADAPTER_DEFAULT - всегда главный адаптер ( под номером 0 ) можно
   вставить цифру
                                    нужного адаптера если в системе их несколько

2. Указатель на структуру параметров экрана  ( см. выше )

Возвращает в случае успеха D3D_OK. Если нет то D3DERR_INVALIDCALL - недействительный вызов.
```

Заполняем только те поля структуры которые нам необходимы:

```
mov d3dpp.Windowed, TRUE    ; Работаем в окошке
mov d3dpp.SwapEffect, D3DSWAPEFFECT_FLIP ; Будем менять буфера
mov eax, d3ddm.Format    ; Берем формат экрана
mov d3dpp.BackBufferFormat, eax    ; Сохраняем в структуре
```

А сейчас делаем самое главное ! Создаем устройство Direct3DDevice8.

Создавать Direct3DDevice в сообщении WM_Create нельзя ! (проверено опытным путем :))
То же самое можно найти в документации по Direct3D8.

CreateDevice мы должны передать hwnd полностью созданного окна на экране которое находится в фокусе.

```
d3d8  CreateDevice, pd3d, D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hwnd , \
          D3DCREATE_HARDWARE_VERTEXPROCESSING, ADDR d3dpp, ADDR pd3dDevice
```

Справка

```
1. Номер видеоадаптера. D3DADAPTER_DEFAULT - всегда главный адаптер ( под номером 0 ) можно
   вставить цифру
                                    нужного адаптера если в системе их несколько.

2. Тип устройства рендеринга.  D3DDEVTYPE_HAL - аппаратный рендеринг. D3DDEVTYPE_REF - программный
   рендеринг.

3. Hwnd - ID нашего окошка.

4. Комбинация флагов. D3DCREATE_FPU_PRESERVE - нам нужна двойная точность вычисления с плавающей
   запятой.
           D3DCREATE_HARDWARE_VERTEXPROCESSING - аппаратная обработка вершин
          D3DCREATE_SOFTWARE_VERTEXPROCESSING - программная обработка вершин
           D3DCREATE_MIXED_VERTEXPROCESSING - аппаратная + программная обработка вершин
           D3DCREATE_MULTITHREADED - многопоточность приложения ( direct3D создает глобальный
            критический раздел это может привести к падению FPS )
           D3DCREATE_PUREDEVICE - не поддерживать Get вызовы. Также не обеспечивать эмуляцию
                 каких- либо сервисов связанных с Vertex processing. Имеется в виду
                 что если устройство не поддерживает vertex processing тогда
              приложение может использовать только post трансформацию вершин.
        D3DCREATE_DISABLE_DRIVER_MANAGEMENT - D3D будет управлять ресурсами вместо драйвера.

Флаги SOFTWARE_VERTEXPROCESSING, HARDWARE_VERTEXPROCESSING и MIXED_VERTEXPROCESSING недопустимы
для совместного использования.

5. Указатель на структуру D3DPRESENT_PARAMETERS. ( см. выше )
6. Адрес переменной куда будет помещен указатель на D3DDevice

Возвращает в случае успеха D3D_OK.
Если нет то D3DERR_INVALIDCALL - недействительный вызов, D3DERR_NOTAVAILABLE - устройством
   требуемое не
поддерживается или D3DERR_OUTOFVIDEOMEMORY - не хватает видео памяти для данной операции.
```

Теперь наше устройство создано и настало время написать функцию Render_Scene !

итак Render_Scene

Для простого примера применим очистку backbuffer`а цветом и показа его на экране:

```
d3dev8 Clear, pd3dDevice, 0 , NULL , D3DCLEAR_TARGET, clearcolor, Zvalue, 0
```

Справка

1. Число структур прямоугольников в массиве. (см. Параметр 2)
2. Указатель на массив структур прямоугольников. Каждый прямоугольник указывает какое место в поверхности
   рендеринга очищать. Если 0 то очищается вся поверхность.
   При этом в первом параметре тоже должен быть 0.

3. Комбинация флагов. D3DCLEAR_STENCIL - очищать стенсель буфер стенсель значением. (см. параметр 6)
   D3DCLEAR_TARGET - очищать поверхность рендеринга цветовым значением. (см.параметр 4)
   D3DCLEAR_ZBUFFER - очищать Z буфер z значением. (см.параметр 5)

4. Цветовое значение. 32bit-ное значение в формате ARGB

5. Z значение. Может быть в диапазоне от 0.0 до 1.0 (для z основного или w основного буфера глубины)
   0.0 - ближняя гараница. 1.0 - дальняя граница.

6. Stencil значение. Может быть в диапазоне от 0 до 2^n-1 где n это глубина бит стенсель буфера.

Возвращает в случае успеха D3D_OK. Если нет то D3DERR_INVALIDCALL,
D3DERR_NOTAVAILABLE или D3DERR_OUTOFVIDEOMEMORY. (расшифровку см. выше)

А теперь все покажем этой функцией

```
d3dev8     Present, pd3dDevice, NULL, NULL, NULL, NULL
```

Справка

1. Указатель на структуру источник типа RECT. Должно быть 0 если поверхности не были созданы с флагами
   D3DSWAPEFFECT_COPY и D3DSWAPEFFECT_COPY_VSYNC.

2. Указатель на структуру назначение типа RECT. Должно быть 0 если поверхности не были созданы с флагами
   D3DSWAPEFFECT_COPY и D3DSWAPEFFECT_COPY_VSYNC.

3. ID окошка куда показывать результат. Если мы указали его ранее в структуре D3DPRESENT_PARAMETERS, то здесь 0
4. Не используется и должен быть 0.

Откомпилировав и запустив это приложение, вы увидите, что оно работает. Но это еще не все. После того как наше приложение будет закрыто, нам необходимо удалить все созданные нами интерфейсы и занятые ими ресурсы. Для этого создадим функцию Destroy_Direct3D.

итак Destroy_Direct3D

```
d3dev8 Release, pd3dDevice ; Метод специально для Direct3DDevice8
d3d8 Release, pd3d  ; Метод специально для Direct3D8
```

На этом позволю себе откланяться. До следующих встреч !

Исходник прилагается (находится в архиве `wasm_files.zip`: `files/0231/dx8masm01.zip`).

PS

В следующем уроке мы рассмотрим запуск приложения в полноэкранном режиме, а также связанную с этим проблему потерянного устройства.

Авторство принадлежит Пономареву Михаилу ака keYMax.
Все вопросы и ругательства слать по адресу mybox@aib.ru

DirectX 8.1 в MASM32: Урок 2

Автор: keYMax

Дата: 2003-03-07

Урок 2. Запуск приложения в полноэкранном режиме.

Привет ! Сегодня мы научимся запускать приложение в полноэкранном режиме. Но сначала о некоторых упущениях с моей стороны :(. В самом первом уроке, где представлен каркас приложения, присутствует ошибка, проявляющаяся в Windows 98 . При запуске окно не появляется на экране и программа висит в памяти (у меня так было, когда решил проверить программу под Win 98). В Windows XP такой проблемы нет. Все из-за того, что я неосторожно обошелся с параметром hCursor в структуре WNDCLASSEX, поместив ID обычной стрелки сразу, а не получая его от Windows в процессе работы приложения. Чтобы ликвидировать это недоразумение, необходимо перед RegisterClassEx вставить следующие строчки:

```
Invoke LoadCursor, NULL, IDC_ARROW
Mov hCursor, eax
```

После этого у меня все заработало :). Можно их и не писать, а занести в hCursor ноль, но тогда в окошке будет курсор в виде часов. Также можно курсор вообще отрубить. Не сомневаюсь, что, если у вас возникла эта проблема, вы сразу разобрались. Однако лучше обьявить об ошибке поздно, чем никогда. Чтобы избежать досадных багов в будущем, я буду проверять работу приложений и в 98 и в XP (к NT и 2000 у меня доступа нет).

Теперь комментарии к уроку номер 1.

Как вы могли заметить, я не проверяю коды возврата при вызове функций Direct3D, кроме тех случаев где это действительно необходимо. Конечно это плохо, если возникнет ошибка, то придется долго искать какая функция не сработала. Решение принято для того, чтобы меньше засорять исходник (вы и сами можете добавить обработку ошибок). Результатом такого шага является или идеальная работа приложения, или его полная неработоспособность при запуске. Аппаратура также бывает разная, и то, что работает на моем ускорителе, может не работать у вас. Особое внимание прошу обратить на CreateDevice и передаваемые ей значения, например, D3DDEVTYPE_HAL, D3DCREATE_SOFTWARE_VERTEXPROCESSING и т.д. Тоже самое относиться и к другим важным функциям. Если видеокарта не поддерживает какие-то параметры, то запуск приложения закончится крахом. В общем, в случае неудачи пробуйте и экспериментируйте, а не ругайте дядю, который написал хрен знает что (сам иногда ругаюсь, если не получается ;)). Специально для таких случаев я и привожу подробную справку.

Хотелось бы заметить еще об одной важной детали: включаемые файлы INC. Начиная с этого урока, я буду включать в исходник только те INC файлы, которые необходимы именно для него. Советую собрать их все в одной папке (у меня это \MASM32\DirectX81\ ) и в исходнике менять мои пути на ваши (это относиться и к LIB файлам). Внутри каждого INC файла есть дата его последней правки. Так что смотрите и своевременно заменяйте их на более свежие (они будут вкладываться в исходник в обязательном порядке).

Ну вот вроде бы ничего не забыл. Переходим к уроку...

Запуск приложения в полноэкранном режиме почти не отличается от запуска в окошке. Все что нам нужно, это заполнить несколько новых параметров в структуре D3DPRESENT_PARAMETERS. Соответственно берем первый урок и после небольших изменений все готово.

В оригинальном хелпе рекомендуется создавать окошко таким размером, какое разрешение мы хотим включить. Т.е. включаем 640x480 - значит создаем окошко 640x480 и т.п. Также его рекомендуется помещать на экран в координаты 0,0.

Для большей наглядности я прописал содержимое структуры D3DPRESENT_PARAMETERS в секции DATA. Описание ее было в предыдущем уроке.

Для нас важно занести в нее следующее:

```
BackBufferWidth dd APP_X    ; Ширина ( разрешение по горизонтали )
BackBufferHeight dd APP_Y   ; Высота ( разрешение по вертикали )
BackBufferFormat dd D3DFMT_X8R8G8B8 ; Формат используемой поверхности
BackBufferCount dd 3    ; число Back буферов.
...
SwapEffect  dd D3DSWAPEFFECT_FLIP  ; Эффект обмена BackBuffer
...
Windowed    dd FALSE  ; 0 - полноэкранный
FullScreen_RefreshRateInHz  dd 100  ; Частота обновления экрана в Hz
FullScreen_PresentationInterval dd 1  ; Показывать не превышая частоту
```

Формат поверхности я взял D3DFMT_X8R8G8B8. Можете использовать любой другой, который нравится. Число BackBuffer'ов равно 3, годится и 2. SwapEffect тоже выбирайте на вкус. В поле Windowed обязательно надо занести 0. В FullScreen_RefreshRateInHz ставите число герц, поддерживаемое данным режимом (главное не переборщить). Ну и PresentationInterval не забудьте.

Далее все как обычно: создаем устройство и работаем.

Вроде бы, на первый взгляд, все отлично. Но стоит нам нажать ALT+TAB, переключаясь на другую задачу, и затем снова перейти к нашему приложению, то возникнет небольшая проблема. Называется она "проблемой потерянного устройства" или LostDevice и решается легко. Всего то надо добавить пару лишних строчек кода.

Пишем в функции Render_Scene:

```
d3dev8   TestCooperativeLevel, pd3dDevice  ; Проверяем кооперацию
cmp    eax, D3DERR_DEVICELOST   ; Выход если устройство потеряно
jne   noreset
ret

noreset:
cmp    eax, D3DERR_DEVICENOTRESET  ; Устройство сброшено или нет
jne   noreset2

d3dev8   Reset, pd3dDevice, ADDR BackBufferWidth ; Сбрасываем
noreset2:
```

Справка

```
TestCooperativeLevel параметров не передается.

Метод возвращает D3D_OK если все прошло успешно.
Если нет, то D3DERR_DEVICELOST - устройство потеряно и не может быть
восстановлено. Рендеринг невозможен.
И D3DERR_DEVICENOTRESET - устройство не может быть сброшено

Reset передается указатель на структуру D3DPRESENT_PARAMETERS ( параметр не может быть нулем )

Метод возвращает D3D_OK если все прошло успешно.
Если нет, то D3DERR_INVALIDCALL, D3DERR_OUTOFVIDEOMEMORY или D3DERR_OUTOFMEMORY
```

Проверяем устройство на кооперацию, если оно потеряно, то мы ничего поделать не можем и выходим из функции. Если устройство не потеряно, то проверяем его на предмет сброшено оно или нет. Если оказывается что оно не сброшено, то тогда сбрасываем его. Вот и все.

Добавив эти строчки, мы можем смело переключаться с задачи на задачу, и при возвращении в наше приложение все будет восстанавливаться и работать дальше !

Напоследок предлагаю вам небольшую задачу.

Данный кусок кода выполняется в функции Render_Scene всегда. Т.е. если у нас приложение выдает, например, 1000 FPS, то проверка потери устройства происходит 1000 раз. Такой расклад никуда не годится, потому что реально устройство оказывается сброшеным только тогда, когда мы нажимаем ALT+TAB и переключаемся на другую задачу. Отсюда вывод: надо сделать так, чтобы проверка осуществлялась вне цикла отрисовки сцены, а конкретно в сообщении, когда приложение получает, например, фокус и только тогда сбрасывать устройство.

Я пробовал такой вариант и он почти работоспособен. Вам его не предлагаю, так как он имеет маленькое неудобство. Если кто-то сможет решить задачу, то я с удовольствием бы взглянул на результат.

Вот и все.

В следующем уроке мы создадим диалог для определения всех доступных режимов экрана, и тогда приложение будет по настоящему правильным.

P.S.

Все еще только начинается ...

Мужики, вы чего такие стеснительные. Не верю, что всем все нравиться. Высказывайте свое мнение. Учту его при дальнейшей работе.

Успехов всем в труде и обороне :)

Все вопросы мыльте сюда mybox@aib.ru

DirectX 8.1 в MASM32: Урок 3

Автор: keYMax

Дата: 2003-03-26

Урок 3. Создаем диалоговое окно для определения всех доступных режимов.

Как определить все доступные полноэкранные режимы и при этом сделать так, чтобы пользователь смог выбирать в каком режиме работать приложению ? В этом нам поможет диалоговое окно.

Если вы уже знаете, что такое диалоговое окно и представляете, как оно выглядит, как его создать и как с ним работать, то все равно советую вам почитать эту статью. Тем же, кто плохо понимает, что это за зверь такой, следует дополнительно посмотреть статьи на эту тему.

Готовы ? Тогда вперед !

Диалоговое окно в нашем приложении можно реализовать двумя путями. Первый путь - это встроить его в само приложение. Второй путь - это реализовать его как самостоятельное приложение. Что выбрать ? Оба варианта имеют свои плюсы и минусы.

Среди плюсов первого варианта то, что оно находится внутри нашего приложения. При этом нам нужно написать не такой уж большой обьем кода. Также все настройки, выбранные пользователем, доступны немедленно. Среди минусов то, что при каждом запуске приложения пользователю приходиться выбирать настройки.

Реализация второго варианта несколько сложнее. Нужно создавать два экзешника. Один для диалогового окна, другой - собственно для приложения. При этом все настройки, произведенные пользователем, придется записывать в файл. Но зато появляется возможность переделывать Setup снова и снова, не трогая при этом приложение, ведь всё равно все настройки сохраняются в файле.

В итоге каждый вариант подходит для определенных задач. Например, для какой - нибудь демки встроенное диалоговое окно самое то. А вот для игрушки - наоборот. Можно также придумать какой нибудь гибрид. Я долго размышлял какой вариант использовать для урока и в конце концов решил сделать оба, чтобы у вас была возможность сравнить. Это потребовало довольно много времени, и как следствие, урок был готов гораздо позже, чем было запланировано. Мы с вами рассмотрим только второй вариант, так как он немного сложнее.

По плану нам требуется: определить все доступные оконные режимы, определить все доступные полоноэкранные режимы, запихать всю эту информацию в диалоговое окно, вывести его на экран, обработать все действия пользователя, сохранить настройки в файле и при выходе из Setup запустить само приложение если требуется. Вроде бы целая куча работы, но на самом деле все гораздо проще.

Начнем с файла ресурсов. Берем любой редактор, пара минут и все готово. Кратко рассмотрим по частям, как он должен выглядеть.

```
#define DS_CENTER            0x00000800L
#define DS_MODALFRAME        0x00000080L
#define WS_EX_TOOLWINDOW     0x00000080L
#define WS_GROUP             0x00020000L
#define WS_EX_STATICEDGE     0x00020000L
#define BS_AUTORADIOBUTTON   0x00000009L
#define BS_FLAT              0x00008000L
#define CBS_DROPDOWNLIST     0x00000003L
#define WS_VSCROLL           0x00200000L
#define WS_TABSTOP           0x00010000L
#define WS_DISABLED          0x08000000L
#define ID_OK                10
```

```
#define ID_CANCEL            11
#define ID_SAVE              12

#define IDC_STATIC           -1

#define IDC_WINDOWED         20
#define IDC_FULLSCREEN       21
#define IDC_SCREENMODESFS    22
#define IDC_SCREENMODESW     23
#define IDC_HERZ             24
```

Здесь у нас константы, в частности, номера от 10 до 24 - это идентификаторы элементов нашего диалогового окна. Они будут нам передаваться в сообщениях от Windows, так мы сможем узнать с каким элементом в данный момент работает пользователь.

```
SetupDialog  DIALOG LOADONCALL MOVEABLE DISCARDABLE 0, 0, 175, 91
             STYLE DS_MODALFRAME | DS_CENTER
             EXSTYLE WS_EX_TOOLWINDOW
             CAPTION " Настройка параметров запуска"
             FONT 8, "Verdana"
```

Здесь имя диалога, его вид и размер, параметры вывода на экран, текст заголовка и используемый шрифт.

```
BEGIN
CONTROL "В окошке",IDC_WINDOWED,"Button",BS_AUTORADIOBUTTON | BS_FLAT | WS_TABSTOP ,12,14,80,14

COMBOBOX IDC_SCREENMODESW,92,14,71,50,CBS_DROPDOWNLIST | WS_VSCROLL | WS_TABSTOP

CONTROL "Полноэкранное",IDC_FULLSCREEN,"Button",BS_AUTORADIOBUTTON | BS_FLAT | WS_TABSTOP,12,30,80,
    12

COMBOBOX IDC_SCREENMODESFS,12,43,71,120,CBS_DROPDOWNLIST | WS_VSCROLL | WS_DISABLED | WS_TABSTOP

LTEXT "Частота в герцах",IDC_STATIC,94,33,65,10

COMBOBOX  IDC_HERZ, 92, 43, 71, 80, CBS_DROPDOWNLIST | WS_VSCROLL | WS_DISABLED | WS_TABSTOP

GROUPBOX  "",  IDC_STATIC, 6,3,163,61

DEFPUSHBUTTON "Запуск", ID_OK,118,69,51,17,0,WS_EX_STATICEDGE
PUSHBUTTON  "Выход", ID_CANCEL,62,69,51,17,0,WS_EX_STATICEDGE
PUSHBUTTON  "Сохранить", ID_SAVE,6,69,51,17,0,WS_EX_STATICEDGE

END
```

Между строками BEGIN и END располагаются строки, описывающие элементы расположенные на диалоговом окне. Здесь определены два RadioButton, три ComboBox, одно текстовое поле, одна рамка и три простых кнопки.

А вот и исходный текст Setup.exe:

```
.686
.MMX
.XMM
.MODEL FLAT, STDCALL
 OPTION CASEMAP:none

INCLUDE  \masm32\include\windows.inc
INCLUDE  \masm32\include\kernel32.inc
INCLUDE  \masm32\include\user32.inc
INCLUDE  \masm32\include\gdi32.inc
INCLUDE  \masm32\include\shell32.inc
INCLUDE  \masm32\DirectX81\d3d8.inc

INCLUDELIB  \masm32\lib\kernel32.lib
INCLUDELIB  \masm32\lib\user32.lib
INCLUDELIB  \masm32\lib\gdi32.lib
INCLUDELIB  \masm32\lib\shell32.lib
INCLUDELIB  \masm32\directx81\d3d8.lib
```

```
SetupDialog  PROTO :DWORD,:DWORD,:DWORD,:DWORD
DW2A         PROTO :DWORD, :DWORD

.DATA ;----------------------------------------

 szDialogName  db  "SetupDialog",0
 szAppName     db  "Tutorial03.exe",0
 szFileName    db  "Settings.ini",0
 szOperation   db  "open",0

 WinModeTable    dw  320,240, 640,480, 800,600, 1024,768, 1280,1024, 1600,1200
 WinModeStrTable dd  OFFSET szMode1 , OFFSET szMode2
                 dd  OFFSET szMode3 , OFFSET szMode4
  dd  OFFSET szMode5 , OFFSET szMode6

 szMode1         db  "320x240",0
 szMode2         db  "640x480",0
 szMode3         db  "800x600",0
 szMode4         db  "1024x768",0
 szMode5         db  "1280x1024",0
 szMode6         db  "1600x1200",0

.DATA? ;----------------------------------------

 pd3d     dd  ?                      ; Указатель на Direct3D
 d3ddm    D3DDISPLAYMODE
        ; Для получения информации о доступных режимах
 d3dpp    D3DPRESENT_PARAMETERS
 ; Структура для создания Direct3DDevice8
 WinStyle dd  ?

 CmbWindow      dd  ?                ; ID ComboBox оконных режимов
 cmbFullScreen  dd  ?                ; ID ComboBox полноэкранных режимов
 cmbHerz        dd  ?                ; ID ComboBox герц

 FullModeTable     dd 50 dup (?)    ; Для хранения доступных полноэкранных режимов
 FullModeBPPTable  db 50 dup (?)   ; Для хранения форматов поверхности.
 FullModeHerzTable db 50 dup (?)    ; Для хранения частоты в Герцах.
 FullModeStrBuffer db 16 dup (?)    ; Буфер для формирования строки

 AppPath   db  260 dup (?)           ; Буфер для пути к Setup.exe
 hfile     dd  ?
 brr       dd  ?

.CODE ;----------------------------------------

Start:

 Invoke  DialogBoxParam,400000h,ADDR szDialogName,NULL,OFFSET SetupDialog,NULL
 Invoke  ExitProcess,0

 Ret

SetupDialog proc hDlg:HWND , iMsg:DWORD , wParam:WPARAM , lParam:LPARAM
;------------------------------------------------------------------------
 mov  eax, iMsg

 cmp  eax, WM_INITDIALOG  ; Обрабатывая это сообщ. надо ОБЯЗАТЕЛЬНО вернуть EAX=1
 je   wmInitDialog

 cmp  eax, WM_COMMAND
 je   wmCommandDialog

 xor  eax, eax
 ret

wmInitDialog:
;--------------

 pushad
 invoke  CheckRadioButton, hDlg , 20 , 21 , 20
```

```
 invoke  GetDlgItem, hDlg , 23
 mov     cmbWindow,  eax
 invoke  GetDlgItem, hDlg , 22
 mov     cmbFullScreen,  eax
 invoke  GetDlgItem, hDlg , 24
 mov     cmbHerz,  eax

 invoke  Direct3DCreate8, D3D_SDK_VERSION
 mov     pd3d, eax

 d3d8    GetAdapterDisplayMode, pd3d, D3DADAPTER_DEFAULT, ADDR d3dpp

 lea     esi, d3dpp.BackBufferWidth
 mov     eax, [esi]
 add     eax, [esi+4]
 mov     ebx, [esi+12]
 mov     [esi+8], ebx
 mov     DWORD PTR [esi+12], 3
 xor     ebx, ebx

; Проверка доступных оконных режимов и заполнение списка

 getNextMode:
 push    ebx
 push    eax
 invoke  SendMessage,cmbWindow,CB_ADDSTRING,0,[WinModeStrTable + ebx*4]
 pop     eax
 pop     ebx
 movzx   ecx, [WinModeTable+ebx*4]
 movzx   edx, [WinModeTable+ebx*4+2]
 add     ecx, edx
 inc     ebx
 cmp     eax, ecx
 ja      getNextMode

; В итоге в ComboBox занесены строки доступных нам оконных режимов
; от минимальных до текущих размеров экрана включительно

; Проверка доступных полноэкранных режимов и заполнение списка
 d3d8    GetAdapterModeCount, pd3d, D3DADAPTER_DEFAULT
 dec     eax

 lea     esi, FullModeTable
 xor     ebx, ebx
 mov     edx, ebx
 mov     ecx, eax

 getMode:
 push    ecx

 push    esi
 push    ebx
 push    edx

 d3d8    EnumAdapterModes, pd3d, D3DADAPTER_DEFAULT, ecx, ADDR d3ddm
 mov     eax, d3ddm.Width1
 mov     ecx, d3ddm.Height
 mov     edi, eax
 add     edi, ecx
 add     edi, d3ddm.Format

 pop     edx
 pop     ebx
 pop     esi

 cmp     edx, edi
 je      noAdd
 mov     edx, d3ddm.RefreshRate
 mov     BYTE PTR [FullModeHerzTable+ebx], dl
 mov     edx, d3ddm.Format
 mov     [FullModeBPPTable+ebx], dl
```

```
 push    edx
 push    edi
 pop     edx
 pop     edi

 mov     [esi+ebx*4], eax
 mov     [esi+ebx*4+2], ecx
 inc     ebx

 push    ebx
 push    esi
 push    edx

; Формирование строки и добавление ее в Combobox

 invoke  DW2A, eax, ADDR FullModeStrBuffer
 mov     BYTE PTR [ebx], 'x'
 inc     ebx
 invoke  DW2A, ecx, ebx
 mov     BYTE PTR [ebx], 'x'
 inc     ebx
 mov     ecx, 16

 cmp     edi, D3DFMT_X8R8G8B8
 jne     nextFormat
 mov     ecx, 32

 nextFormat:
 cmp     edi, D3DFMT_A8R8G8B8
 jne     nextFormat2
 mov     ecx, 32

 nextFormat2:
 cmp     edi, D3DFMT_R8G8B8
 jne     exitCompare
 mov     ecx, 24

 exitCompare:
 invoke  DW2A, ecx , ebx
 mov     BYTE PTR [ebx], 0
 invoke  SendMessage, cmbFullScreen , CB_ADDSTRING , 0 , ADDR FullModeStrBuffer

 pop     edx
 pop     esi
 pop     ebx

 noAdd:
 pop     ecx
 dec     ecx
 test    ecx, ecx
 jne     getMode

; В итоге в ComboBox занесены строки доступных нам полноэкранных режимов
; с разными форматами поверхности и самым высоким числом Герц для каждого режима

 invoke  SendMessage, cmbWindow , CB_SETCURSEL , 0 , 0
 invoke  SendMessage, cmbFullScreen , CB_SETCURSEL , 0 , 0
 invoke  SendMessage, hDlg, WM_COMMAND , 22 + CBN_SELCHANGE shl 16, 0
 popad

 xor     eax, eax
 inc     eax
 ret

 wmCommandDialog:
;----------------
 movzx   eax, WORD PTR wParam ; Здесь нам нужно получить HIWORD и LOWORD из wParam
 mov     ecx, wParam          ; HIWORD - код уведомления от элемента диалога
                              ; LOWORD - ID элемента диалога
 xor     ebx, ebx
 cmp     eax, 10              ; Получаем если нажали кнопку Запуск
```

```
je   cmStartProgram

cmp    eax, 12              ; Получаем если нажали кнопку Сохранить
je   cmSaveData

cmp    eax, 11              ; Получаем если нажали кнопку Закрыть
je   cmCloseProgram

cmp    eax, 20              ; Получаем если нажали RadioButton "В окошке"
je   cmRadioWin

cmp    eax, 21              ; Получаем если нажали RadioButton "Полноэкранное"
je   cmRadioFull

cmp    ecx, 22 + CBN_SELCHANGE shl 16
je   cmHerz

xor    eax, eax
ret

  cmRadioFull:

  inc     ebx

  cmRadioWin:

  invoke  EnableWindow, cmbFullScreen , ebx
  invoke  EnableWindow, cmbHerz , ebx
  xor     ebx, 1
  invoke  EnableWindow, cmbWindow , ebx

  xor     eax, eax
  ret

  cmHerz:

  invoke  SendMessage, cmbHerz , CB_RESETCONTENT , 0 , 0
  invoke  SendMessage, cmbFullScreen , CB_GETCURSEL , 0 , 0
  movzx   ebx, BYTE PTR [FullModeHerzTable+eax]
  invoke  DW2A, ebx , ADDR FullModeStrBuffer
  invoke  SendMessage, cmbHerz , CB_ADDSTRING , 0 , ADDR FullModeStrBuffer
  invoke  SendMessage, cmbHerz , CB_SETCURSEL , 0 , 0

  xor     eax, eax
  ret

  cmSaveData:

  invoke  IsDlgButtonChecked, hDlg , 20
  test    eax, eax
  je      EnableFull

  invoke  SendMessage, cmbWindow , CB_GETCURSEL , 0 , 0
  lea     esi, WinModeTable
  shl     eax, 2
  add     esi, eax
  xor     ecx, ecx
  mov     edx, ecx
  push    [esi]
  pop     cx
  pop     dx
  mov     d3dpp.BackBufferWidth, ecx
  mov     d3dpp.BackBufferHeight, edx
  mov     d3dpp.Windowed, 1
  mov     WinStyle, WS_SYSMENU
  jmp     Create

  EnableFull:

  invoke  SendMessage, cmbFullScreen , CB_GETCURSEL , 0 , 0
  movzx   ebx, BYTE PTR [FullModeHerzTable+eax]
```

```
    mov     d3dpp.FullScreen_RefreshRateInHz, ebx
    movzx   ebx, BYTE PTR [FullModeBPPTable+eax]
    mov     d3dpp.BackBufferFormat, ebx

    lea     esi, FullModeTable
    shl     eax, 2
    add     esi, eax
    push    [esi]
    xor     ecx, ecx
    mov     edx, ecx
    mov     d3dpp.Windowed, ecx
    pop     cx
    pop     dx
    mov     d3dpp.BackBufferWidth, ecx
    mov     d3dpp.BackBufferHeight, edx
    mov     WinStyle, WS_POPUP

    Create:

    mov     d3dpp.SwapEffect, D3DSWAPEFFECT_FLIP

    invoke  CreateFile , ADDR szFileName , GENERIC_WRITE, NULL, \
            NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL
    mov     hfile, eax

    invoke  WriteFile , hfile , ADDR d3dpp , 56 , ADDR brr , NULL
    invoke  CloseHandle , hfile

    xor     eax, eax
    ret

    cmStartProgram:

    invoke  EndDialog, hDlg , 0
    d3d8    Release, pd3d

    invoke  GetModuleFileName , 0 , ADDR AppPath , 260

    lea     edi,AppPath
    lea     edi,[edi+eax-6]
    @@:
    mov     al,[edi]
    dec     edi
    cmp     al,'\'
    jne     @B
    inc     edi
    inc     edi
    cld
    lea     esi, szAppName
    mov     ecx, SIZEOF szAppName
    shr     ecx, 2
    rep     movsd
    mov     ecx, [SIZEOF szAppName ]
    and     ecx, 3
    rep     movsb

    invoke  ShellExecute , NULL , ADDR szOperation , ADDR AppPath , \
            NULL , NULL , SW_SHOWNORMAL
    xor     eax, eax
    ret

    cmCloseProgram:

    invoke  EndDialog, hDlg , 0
    d3d8    Release, pd3d

    xor     eax, eax
    ret

SetupDialog endp
```

Еще после SetupDialog находится текст функции DW2A. Я не стал ее тут приводить, т.к. она является слегка измененной dw2a из библиотеки masm32. Интересующиеся всегда смогут посмотреть ее в исходнике.

Теперь все разберем по косточкам.

```
WinModeTable    dw  320,240, 640,480, 800,600, 1024,768, 1280,1024, 1600,1200
WinModeStrTable dd  OFFSET szMode1 , OFFSET szMode2
                dd  OFFSET szMode3 , OFFSET szMode4
 dd  OFFSET szMode5 , OFFSET szMode6

szMode1         db  "320x240",0
szMode2         db  "640x480",0
szMode3         db  "800x600",0
szMode4         db  "1024x768",0
szMode5         db  "1280x1024",0
szMode6         db  "1600x1200",0
```

Массив WinModeTable содержит размеры шести определенных нами окошек. Он также нужен нам для сравнения с текущими размерами рабочего стола. Массив WinModeStrTable содержит указатели на текстовые строки. Эти строки впоследствии будут занесены в ComboBox оконных режимов.

```
FullModeTable     dd 50 dup (?)    ; Для хранения доступных полноэкранных режимов
FullModeBPPTable  db 50 dup (?)   ; Для хранения форматов поверхности.
FullModeHerzTable db 50 dup (?)    ; Для хранения частоты в Герцах.
FullModeStrBuffer db 16 dup (?)    ; Буфер для формирования строки
```

В массиве FullModeTable число 50 является количеством всех полноэкранных режимов, которые мы можем поместить. Я, на всякий случай, сделал его с запасом. Вообще, когда мы спрашиваем Direct3D о количестве доступных режимов он возвращает нам огромное их число, куда входят режимы с одинаковым разрешением, но с разным числом герц и форматом поверхности. Например, на моей видюхе он выдает 90 доступных режимов. Чтобы не возиться с разной частотой я решил определять только те режимы, в которых число Герц максимально. В результате мы ничего не теряем, т.к. при выборе настроек включать частоту в 60, 70 и т.п. Герц все равно никто не будет. Взяв на вооружение этот факт, мы сокращаем число режимов до двух - трех десятков (Примечание: В Win98 Direct3D выдает вместо числа Герц ноль, поэтому, если занести в структуру D3DPRESENT_PARAMETERS определенную частоту, а не ноль, то создать устройство не удастся). В массиве FullModeStrBuffer мы будем формировать строку вида: 1024x768x32 и затем заносить ее в ComboBox.

Invoke DialogBoxParam,400000h,ADDR szDialogName,NULL,OFFSET SetupDialog,NULL

Наш диалог из ресурсов вызываем вот такой строкой где 400000h - это hInstance.

```
wmInitDialog:
 ;--------------

  pushad
  invoke  CheckRadioButton, hDlg , 20 , 21 , 20
  invoke  GetDlgItem, hDlg , 23
  mov     cmbWindow,  eax
  invoke  GetDlgItem, hDlg , 22
  mov     cmbFullScreen,  eax
  invoke  GetDlgItem, hDlg , 24
  mov     cmbHerz,  eax
```

Ну вот добрались до самого главного: определения доступных режимов. Вначале, устанавливаем отметку на RadioButton "В окошке" и снимаем с "Полноэкранное". Затем получаем ID всех наших ComboBox'ов. (Примечание: При выходе из обработки сообщения INIT_DIALOG, когда мы уже возвращаем EAX=1 и передаем управление системе, в Win98 происходила ошибка. На мой взгляд, причина кроется в каких - то регистрах, которые мы изменили в процессе работы. Используя PUSHAD при входе и POPAD при выходе, этот баг можно ликвидировать. Что касается XP, то ему,

похоже, до лампочки есть PUSHAD, POPAD или нет.)

```
    invoke  Direct3DCreate8, D3D_SDK_VERSION
    mov     pd3d, eax

    d3d8    GetAdapterDisplayMode, pd3d, D3DADAPTER_DEFAULT, ADDR d3dpp

    lea     esi, d3dpp.BackBufferWidth
    mov     eax, [esi]
    add     eax, [esi+4]
    mov     ebx, [esi+12]
    mov     [esi+8], ebx
    mov     DWORD PTR [esi+12], 3
    xor     ebx, ebx

  ; Проверка доступных оконных режимов и заполнение списка

    getNextMode:
    push    ebx
    push    eax
    invoke  SendMessage,cmbWindow,CB_ADDSTRING,0,[WinModeStrTable + ebx*4]
    pop     eax
    pop     ebx
    movzx   ecx, [WinModeTable+ebx*4]
    movzx   edx, [WinModeTable+ebx*4+2]
    add     ecx, edx
    inc     ebx
    cmp     eax, ecx
    ja      getNextMode
```

Создаем Direct3D, получаем текущие параметры рабочего стола посредством GetAdapterDisplayMode. Так как мы помещаем данные в D3DPRESENT_PARAMETERS, нам придется провести небольшую манипуляцию, занося формат поверхности с четвертого поля структуры на третье. В четвертое поле заносим число BackBuffer равное трем. Сделав это, часть D3DPRESENT_PARAMETERS мы уже сформировали. Далее идет цикл на сравнение суммы текущих ширины и высоты с суммой размеров записанных нами в WinModeTable. Одновременно с этим заполняем ComboBox.

Справка

```
GetAdapterDisplayMode передается два параметра.

1. Номер адаптера. D3DADAPTER_DEFAULT - адаптер по умолчанию.
2. Адрес структуры D3DDISPLAYMODE. ( См. предыдущие уроки )

Метод возвращает D3D_OK и заполняет структуру данными если все прошло успешно.
Если нет, то D3DERR_INVALIDCALL - недействительный вызов.
```

```
    d3d8    GetAdapterModeCount, pd3d, D3DADAPTER_DEFAULT
    dec     eax
```

Далее получаем количество всех доступных полноэкранных режимов и уменьшаем его на единицу, т.к. нумерация начинается с нуля.

Справка

```
GetAdapterModeCount передается только номер адаптера.

Метод возвращает кол-во всех доступных режимов если все прошло успешно.
Если нет, то ноль.
```

```
    lea     esi, FullModeTable
    xor     ebx, ebx
    mov     edx, ebx
    mov     ecx, eax

    getMode:
    push    ecx
    push    esi
```

```
push    ebx
push    edx

d3d8    EnumAdapterModes, pd3d, D3DADAPTER_DEFAULT, ecx, ADDR d3ddm
mov     eax, d3ddm.Width1
mov     ecx, d3ddm.Height
mov     edi, eax
add     edi, ecx
add     edi, d3ddm.Format

pop     edx
pop     ebx
pop     esi

cmp     edx, edi
je      noAdd
mov     edx, d3ddm.RefreshRate
mov     BYTE PTR [FullModeHerzTable+ebx], dl
mov     edx, d3ddm.Format
mov     [FullModeBPPTable+ebx], dl
push    edx
push    edi
pop     edx
pop     edi

mov     [esi+ebx*4], eax
mov     [esi+ebx*4+2], ecx
inc     ebx
```

С загрузки адреса массива начинается цикл сравнения и заполнения его доступными полноэкранными режимами. Каждый раз, уменьшая номер режима и вызывая EnumAdapterModes, мы спрашиваем данные у Direct3D. Затем берем полученные ширину и высоту, складываем их и прибавляем к этой сумме формат поверхности. Сравниваем с предыдущим значением суммы и, если таковой у нас нет (это значит новый режим), заносим всю информацию по разным массивам, и начинаем формировать строку в буфере, чтобы ее можно было занести в Combobox. Соответственно, если эта сумма есть, то ничего заполнять нам не нужно.

Справка

```
EnumAdapterModes передается три параметра.

1. Номер адаптера.
2. Номер режима. От 0 и выше.
3. Адрес структуры D3DDISPLAYMODE

Метод возвращает D3D_OK и заполняет структуру данными если все прошло успешно.
Если нет, то D3DERR_INVALIDCALL - недействительный вызов.
```

```
push    ebx
push    esi
push    edx

; Формирование строки и добавление ее в Combobox

invoke  DW2A, eax, ADDR FullModeStrBuffer
mov     BYTE PTR [ebx], 'x'
inc     ebx
invoke  DW2A, ecx, ebx
mov     BYTE PTR [ebx], 'x'
inc     ebx
mov     ecx, 16

cmp     edi, D3DFMT_X8R8G8B8
jne     nextFormat
mov     ecx, 32

nextFormat:
cmp     edi, D3DFMT_A8R8G8B8
jne     nextFormat2
```

```
mov     ecx, 32

nextFormat2:
cmp     edi, D3DFMT_R8G8B8
jne     exitCompare
mov     ecx, 24

exitCompare:
invoke  DW2A, ecx , ebx
mov     BYTE PTR [ebx], 0
invoke  SendMessage, cmbFullScreen , CB_ADDSTRING , 0 , ADDR FullModeStrBuffer

pop     edx
pop     esi
pop     ebx

noAdd:
pop     ecx
dec     ecx
test    ecx, ecx
jne     getMode

; В итоге в ComboBox занесены строки доступных нам полноэкранных режимов
; с разными форматами поверхности и самым высоким числом Герц для каждого режима
```

Здесь у нас идет формирование строки. Функцией DW2A преобразуем значения в текст, добавляем иксы. Проверяем ID формата и то что получилось добавляем в качестве окончания строки. Теперь нам только осталось занести эту строку туда куда нужно и все. Это мы и делаем, посылая сообщение ComboBox. А далее снова на начало цикла. На первый взгляд все отлично, но есть здесь одно маленькое но - проверка на формат. Осуществляя сравнение с 24 и 32 битным форматом мы считаем все другие форматы 16-битными. Как мне кажется, это не столь важно. Сколько я ни запускал программу на разных видюхах, будь то ATI или NVIDIA они мне выдавали только 2 формата: D3DFMT_A8R8G8B8 и D3DFMT_R5G6B5 (вроде эти :)). Глубже лезть и выяснять, почему такое есть, мне неохота.

```
invoke  SendMessage, cmbWindow , CB_SETCURSEL , 0 , 0
invoke  SendMessage, cmbFullScreen , CB_SETCURSEL , 0 , 0
invoke  SendMessage, hDlg, WM_COMMAND , 22 + CBN_SELCHANGE shl 16, 0
```

Последними командами устанавливаем выбранными первые строки в каждом из ComboBox и обработка WM_INITDIALOG завершена. Но впереди не менее увлекательное занятие: сделать так, чтобы все работало как единое целое. Приступаем к рассмотрению обработки события WM_COMMAND. В нем мы проверяем ID всех элементов и, в зависимости от результата, переходим на нужную метку.

```
cmRadioFull:

inc     ebx

cmRadioWin:

invoke  EnableWindow, cmbFullScreen , ebx
invoke  EnableWindow, cmbHerz , ebx
xor     ebx, 1
invoke  EnableWindow, cmbWindow , ebx

xor     eax, eax
ret
```

Здесь у нас идет включение одних ComboBox и выключение других. Срабатывает это дело только, когда тыкнуть мышкой на одну из RadioButton.

```
cmHerz:

invoke  SendMessage, cmbHerz , CB_RESETCONTENT , 0 , 0
invoke  SendMessage, cmbFullScreen , CB_GETCURSEL , 0 , 0
```

```
movzx   ebx, BYTE PTR [FullModeHerzTable+eax]
invoke  DW2A, ebx , ADDR FullModeStrBuffer
invoke  SendMessage, cmbHerz , CB_ADDSTRING , 0 , ADDR FullModeStrBuffer
invoke  SendMessage, cmbHerz , CB_SETCURSEL , 0 , 0

xor     eax, eax
ret
```

А это срабатывает, когда нам нужно обновить содержимое ComboBox, содержащего информацию о частоте. Мы все стираем, берем номер выбранного элемента в ComboBox, содержащего доступные полноэкранные режимы, и по нему получаем из массива частоту. Преобразуем ее к удобному варианту и заносим в ComboBox.

```
cmSaveData:

invoke  IsDlgButtonChecked, hDlg , 20
test    eax, eax
je      EnableFull

invoke  SendMessage, cmbWindow , CB_GETCURSEL , 0 , 0
lea     esi, WinModeTable
shl     eax, 2
add     esi, eax
xor     ecx, ecx
mov     edx, ecx
push    [esi]
pop     cx
pop     dx
mov     d3dpp.BackBufferWidth, ecx
mov     d3dpp.BackBufferHeight, edx
mov     d3dpp.Windowed, 1
mov     WinStyle, WS_SYSMENU
jmp     Create

EnableFull:

invoke  SendMessage, cmbFullScreen , CB_GETCURSEL , 0 , 0
movzx   ebx, BYTE PTR [FullModeHerzTable+eax]
mov     d3dpp.FullScreen_RefreshRateInHz, ebx
movzx   ebx, BYTE PTR [FullModeBPPTable+eax]
mov     d3dpp.BackBufferFormat, ebx

lea     esi, FullModeTable
shl     eax, 2
add     esi, eax
push    [esi]
xor     ecx, ecx
mov     edx, ecx
mov     d3dpp.Windowed, ecx
pop     cx
pop     dx
mov     d3dpp.BackBufferWidth, ecx
mov     d3dpp.BackBufferHeight, edx
mov     WinStyle, WS_POPUP

Create:

mov     d3dpp.SwapEffect, D3DSWAPEFFECT_FLIP

invoke  CreateFile , ADDR szFileName , GENERIC_WRITE, NULL, \
        NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL
mov     hfile, eax

invoke  WriteFile , hfile , ADDR d3dpp , 56 , ADDR brr , NULL
invoke  CloseHandle , hfile

xor     eax, eax
ret
```

Окончательное формирование D3DPRESENT_PARAMETERS, плюс сохранение всех настроек в файле. Поздравляю 90% работы позади. Сюда мы попадаем, если пользователь нажал кнопку "Сохранить". Ну значит проверяем какая RadioButton выделена и идем по нужному пути. Если "В окошке", то поле Windowed равно 1 и стиль окна WS_SYSMENU. Если "Полноэкранное", то берем номер выбранной строки в ComboBox полноэкранных режимов и уже от этого пляшем. Хватаем из массивов размеры, формат поверхности, частоту и в структуру все укладываем. Поле Windowed равно нулю, а стиль окна будет WS_POPUP. Окончательный штришок в виде SwapEffect и можно все записывать в файл.

Если мы хотим чтобы из диалога вызывалась еще и программа, то нужно написать следующее.

```
cmStartProgram:

invoke  EndDialog, hDlg , 0
d3d8    Release, pd3d

invoke  GetModuleFileName , 0 , ADDR AppPath , 260

lea     edi,AppPath
lea     edi,[edi+eax-6]
@@:
mov     al,[edi]
dec     edi
cmp     al,'\'
jne     @B
inc     edi
inc     edi
cld
lea     esi, szAppName
mov     ecx, SIZEOF szAppName
shr     ecx, 2
rep     movsd
mov     ecx, [SIZEOF szAppName ]
and     ecx, 3
rep     movsb

invoke  ShellExecute , NULL , ADDR szOperation , ADDR AppPath , \
        NULL , NULL , SW_SHOWNORMAL
xor     eax, eax
ret
```

Закрываем диалог. Освобождаем все связанное с Direct3D. Затем получаем путь по которому лежит наш Setup посредством GetModuleFileName. На всякий случай размер буфера под путь равен 260 байт (если не ошибаюсь это максимальный размер пути в окошках). Вот мы получили путь, но в нем присутствует и имя нашего Setup. Как его убрать ? Делаем цикл для проверки на '\' ,начиная от конца пути. Найдя его, копируем на место следующего за ним имя файла, который хотим запустить. А дальше все просто. ShellExecute с командой open в качестве параметра, и приложение запущено по нашему приказу, а работа самого Setup завершена.

Осталось только написать текст самого приложения. В нем реализовать загрузку сохраненных настроек и на их основе запускать Direct3D. Описывать здесь я это не буду, лучше гляньте исходник.

Ну все, пора закругляться, а то и так много понаписал, да и чего - то под конец статьи совсем туго соображать стал.

PS: В следующей статье ждите полноценное трехмерное приложение с матрицами, векторами, полигонами и прочей математикой.

Да и не забудьте глянуть в папку Include там лежат файлы d3d.inc и d3dcaps.inc окончательной версии.

Исходник обоих вариантов прилагается (1, 2 (находится в архиве `wasm_files.zip`: `files/0234/dx8masm03b.zip`)).

Все вопросы мыльте сюда mybox@aib.ru

DirectX 8.1 в MASM32: Урок 4

Автор: keYMax

Дата: 2003-05-19

Урок 4. Первое полноценное трехмерное приложение.

Основы трехмерной графики. Как это все работает. Я расскажу вам как устроен механизм визуализации в DirectX, и вы научитесь использовать в своей работе простые полигоны.

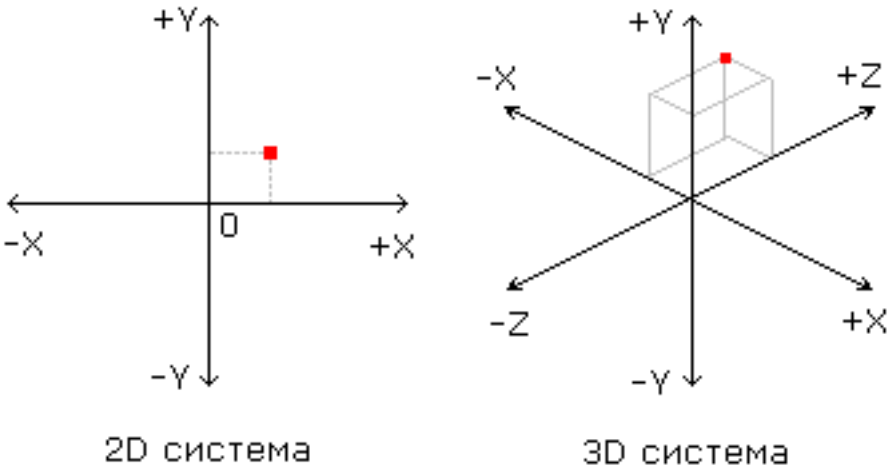
1. Система координат
2. Формат вершин
3. Полигоны
4. Что такое сцена. Как Direct3D работает с трехмерным миром.
5. Камера
6. Z буфер
7. Все знания применим на деле !
    1. Об используемых библиотеках
    2. Новое в диалоге
    3. Новое в приложении
    4. Функция Init_Direct3D
    5. Функция Init_Scene
    6. Функция Set_Render_Parameters
    7. Функция Render_Scene
    8. Функция Animate_Scene
    9. Функция Destroy_Direct3D
8. Послесловие

Предисловие

Как вы знаете, вся трехмерная графика основана на математике. "Линейная алгебра", "Векторная алгебра и начала анализа", "Начертательная геометрия" и еще много чего другого необходимо усвоить прежде чем приступать к успешной работе с 3D. Конечно, это не так интересно, но все равно придется этим заниматься. С вашей стороны необходимо всего лишь желание, а я в свою очередь постараюсь рассказать все как можно проще и доступней на примерах и картинках. Кому не терпится перейти сразу к делу начинайте с пункта 7.

1. Система координат

Начнем с самых основ. Загадочная буква "D" в словах 2D и 3D расшифровывается как Dimension - измерение. Соответственно 2D - это 2 измерения, а 3D - три. 2D представляет собой обычную систему координат на плоскости, которая знакома всем еще со школы. А что представляет собой 3D ? Это та же система координат, только к ней добавляется еще одно измерение обозначаемое как Z. На рисунке изображены обе системы.



Из рисунка видно, что новая координата Z отвечает за то, насколько далеко находится от нас какой - либо обьект. Обе системы координат (2D и 3D) существуют в двух разных вариантах - левосторонняя и правосторонняя. Разница между ними в направлении отсчета. В левосторонней X и Y увеличиваются слева направо, а в правосторонней справа налево. Соответственно координата Z тоже меняет свое направление. На рисунке изображены как раз левосторонние системы. Direct3D позволяет использовать ту, которая вам нравится. Я, как все нормальные люди, буду использовать стандартную левостороннюю систему.

2. Формат вершин

Точка, точка, запятая - вышла рожица кривая. Самое простое, что существует в геометрии - это точка. В трехмерном пространстве точка трактуется как вершина - Vertex. В Direct3D у вершины помимо координат могут быть дополнительно какие - либо свойства. Координаты совместно со свойствами хранятся в одном месте и называются форматом вершины - Vertex Format. Direct3D позволяет задавать различный формат вершины, так называемый FVF - Flexible Vertex Format - гибкий формат вершин. Выбор формата зависит от тех задач, которые вы перед собой ставите. Он может состоять из нескольких флагов. Каждый флаг в отдельности описывает каким свойством будет обладать наша вершина. Допускается объединение сколько угодно таких флагов, если они не противоречат друг другу. Соответственно не забывайте, что чем больше флагов вы используете в формате вершины, тем больший размер она будет занимать в памяти.

Например, часто используемый формат содержит:

```
D3DFVF_XYZ       - координаты вершины
D3DFVF_NORMAL    - вектор для расчета освещения
D3DFVF_DIFFUSE   - цвет для закраски
D3DFVF_TEX2      - координаты текстуры

Все это занимает: (4+4+4)+(4+4+4)+4+(4+4) = 36 байт.
```

Теперь вы видите, что всего лишь на одну вершину при таком формате будет затрачено не так и мало памяти.

Хорошо. Вы выбрали формат, который подходит для ваших целей. Как его описать? Делается это так: D3DFVF_CUSTOMVERTEX equ D3DFVF_DIFFUSE or D3DFVF_XYZ or D3DFVF_NORMAL и т.д

D3DFVF_CUSTOMVERTEX - это простая константа, определенная вами, которая позволяет объединять все нужные свойства для вершины. Далее D3DFVF_CUSTOMVERTEX можно уже указывать в качестве параметра в нужной нам функции. Можно D3DFVF_CUSTOMVERTEX и не использовать, но тогда в каждой функции, где необходимо указывать формат вершины, нужно будет перечислять все флаги, что не очень удобно. Пока остановимся на этом. Далее в тексте

статьи мы еще поговорим о вершинах.

```
Справка

  Все возможные флаги для формата вершины:
  ----------------------------------------
  1. D3DFVF_DIFFUSE  - вершина содержит компоненту цвета.
  2. D3DFVF_NORMAL   - вершина содержит нормаль.
                       Нельзя применять с D3DFVF_XYZRHW
  3. D3DFVF_PSIZE    - вершина специфичного формата.
  4. D3DFVF_SPECULAR - вершина содержит Specular компоненту.
  5. D3DFVF_XYZ      - нетрансформированная вершина содержит позицию.
                       Нельзя использовать вместе с D3DFVF_XYZRHW
  6. D3DFVF_XYZRHW   - трансформированная вершина содержит позицию
                       Нельзя использовать вместе с D3DFVF_XYZ и D3DFVF_NORMAL
  7. D3DFVF_XYZB1    - содержит позицию и значение Beta.
             ...       Beta используется при мультиматричных блендинг
     D3DFVF_XYZB5      ( смешивание ) операциях.
                       На данный момент поддерживается три значения Beta
                       и четыре блендинг матрицы.

  Все возможные флаги, относящиеся к текстуре:
  ---------------------------------------------
  1. D3DFVF_TEX0           - вершина содержит "n" число текстурных координат
     ...
     D3DFVF_TEX8

  2. D3DFVF_TEXTUREFORMAT1 - указывает какой размерности текстурная координата
     ...                     Например: D3DFVF_TEXTUREFORMAT1 - одномерная,
     D3DFVF_TEXTUREFORMAT4   D3DFVF_TEXTUREFORMAT2 - двумерная и т.д.
                             На данный момент эта константа редко используется
  Значения масок:
  ---------------
  1. D3DFVF_POSITION_MASK  - битовая маска позиций
  2. D3DFVF_RESERVED0      - маска зарезервированных битов в формате вершины.
     и                      На данный момент не используется
     D3DFVF_RESERVED2
  3. D3DFVF_TEXCOUNT_MASK  - битовая маска для текстурных флагов.

  Прочее:
  -------
  1. D3DFVF_LASTBETA_UBYTE4 - когда используется индексное вершинное смешение
                              ( vertex blending ) и фиксирована функция FVF
                              вертексных шейдеров, обязательно необходимо
                              указать этот флаг для вертексного шейдера
  2. D3DFVF_TEXCOUNT_SHIFT  - число бит, на которое сдвинуто значение
                              идентифицирующее число текстурных координат для
                              вершины.
  Несколько примеров:
  -------------------
  1. Легкая нетрансформированная и нетекстурированная вершина.
     Закраска по Гуро.

     D3DFVF_CUSTOMVERTEX  equ  D3DFVF_DIFFUSE or D3DFVF_XYZ

  2. Легкая нетрансформированная и нетекстурированная вершина.
     Закраска по Гуро. Используется освещение.

     D3DFVF_CUSTOMVERTEX  equ  D3DFVF_DIFFUSE or D3DFVF_XYZ or D3DFVF_NORMAL

  3. Нетрансформированная вершина использующая текстуру.
     Освещение отсутствует.

     D3DFVF_CUSTOMVERTEX  equ D3DFVF_XYZ or D3DFVF_TEX2

  4. Трансформируемая вершина с RHW частью, использующая текстуру

     D3DFVF_CUSTOMVERTEX  equ D3DFVF_XYZRHW or D3DFVF_TEX2

  5. Тяжелая вершина с компонентой цвета, координатами текстуры, использующая
     освещение.
```

```
D3DFVF_CUSTOMVERTEX equ D3DFVF_XYZ or D3DFVF_NORMAL or D3DFVF_DIFFUSE or /
                        D3DFVF_SPECULAR or D3DFVF_TEX2
```

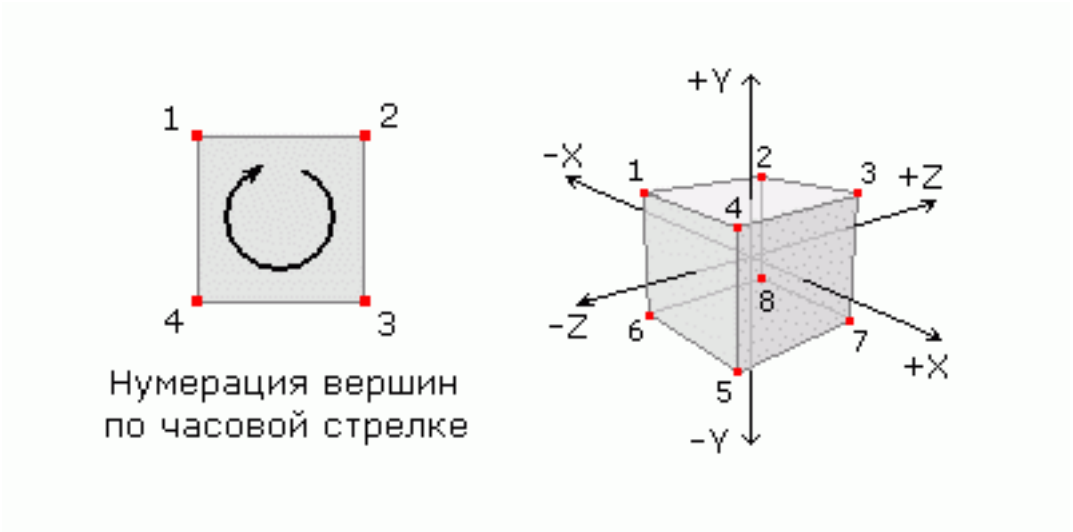
3. Полигоны

Формат вершины уже не является для вас тайной. Ну а как же составить из разрозненных вершин какой либо обьект ? Все объекты состоят из треугольников, скажете вы и будете правы. Конечно в Direct3D можно использовать и другие фигуры, число углов у которых больше чем три. Это квадраты и многоугольники. Но, так как аппаратная часть ускорителя оптимизирована на работу с треугольниками, то перед отрисовкой ваших квадратов они все равно будут разбиты на треугольники, так что сэкономив на простоте описания обьекта, можно проиграть в скорости. Как и что лучше использовать, решать вам.

А почему выбрали именно треугольник ? Потому что он является самой простой фигурой и всегда выпуклый. А раз он выпуклый, то упрощаются различные расчеты при работе с ним (пересечение прямой, лежит ли точка внутри него и т.д.) Вы это поймете сами, когда еще более углубитесь в работу с 3D.

Все фигуры в трехмерной графике называют "полигонами" - если перевести буквально, то это означает многоугольник. Но так как используют практически одни треугольники, то и название полигон стало синонимом слова треугольник. И когда вы видите в какой - либо статье про новую игрушку, что в модели монстра используется 5000 полигонов, то это значит, что он состоит из 5000 треугольников.

Сейчас давайте рассмотрим, как описываются полигоны. Для быстроты просчета сцены (это определение я раскрою чуть дальше) в Direct3D (да и в любом другом 3D API) применяется схема, когда невидимые полигоны отбрасываются и просчитываются только те, которые мы с вами видим. Как же Direct3D может узнать какие полигоны рисовать, а какие нет? Все дело в том как они заданы. И огромную роль в этом играет порядок следования вершин в описании полигона. Догадались? Правильно. Они должны быть заданы либо по часовой стрелке, либо против. Стандартом является то, когда вершины, составляющие полигон, заданы по часовой стрелке. В Direct3D этот режим включен по умолчанию. Т.е. он будет отрисовывать только эти полигоны, а все другие, где вершины описаны против часовой стрелки, отбрасывать. Нижеследующий рисунок поможет вам прояснить все моменты.



Чтобы было еще более понятно, разберем пример. У нас есть куб. Он повернут к нам одной гранью. Начинаем писать вершины этой грани по часовой стрелке. Начинать можно с любой. Когда мы написали эти четыре вершины друг за дружкой, то поворачиваем к себе куб новой гранью и опять пишем вершины по часовой стрелке и т.д. пока все грани не будут описаны. Если теперь внимательно подумать и посмотреть в каком порядке будут находиться вершины той грани, которая

повернута от нас, то окажется, что они будут расположены против часовой стрелки (если смотреть на эту грань сквозь куб). Вот и весь фокус. Соответственно раз они против часовой стрелки и грань куба нам действительно не видно, то и рисовать ее не надо, что и делает с успехом Direct3D (с вашей помощью, конечно). Если вы собираетесь вручную описывать обьекты для дальнейшего использования в программе, то от этой процедуры вам не избавиться как бы этого ни хотелось. Ну а если использовать 3D редактор, то он все сделает за вас автоматически и опишет каждый полигон в нужном порядке. Также довожу до вашего сведения, что в Direct3D в любой момент можно поменять условие, по которому будет определяться какие полигоны отбрасывать. Те которые по часовой стрелке или против. В некоторых случаях это действительно необходимо, но об этом как - нибудь в другой раз.

Теперь когда вы знаете о том, что такое полигоны и как их нужно описывать, пора переходить к следующей части.

4. Что такое сцена. Как Direct3D работает с трехмерным миром.

В трехмерной графике есть такое понятие как сцена. Вы были в театре и видели там сцену, на которой выступают актеры. Так вот здесь это тоже самое. На ней также происходят различные действия. Все обьекты, с которыми вы работаете в данный момент, манипулируете и пытаетесь отрисовать, и представляют собой в совокупности сцену. Простым языком, сцена - это то, что мы в данный момент видим на экране.

Допустим, мы имеем обьект из полигонов на сцене. Как мы можем им манипулировать, перемещать, вращать вокруг оси и т.д? Не забудьте при этом, что все присутствующее на сцене располагается в трехмерной системе координат. Для перемещения вдоль какой - либо оси достаточно изменить координаты каждой из вершин в ту или иную сторону. Для поворота все уже намного сложнее. Тут нужно применять формулы, основаные на синусе и косинусе. Все они уже давно написаны математиками и с блеском работают. Главное их неудобство заключается в громоздкости и в том, что для поворота вокруг нескольких осей сразу, нужно применять эти формулы по очереди для каждой оси в отдельности. Результат может оказаться не таким, какой мы ожидали (думаю, то, что они применяются для каждой вершины отдельно, обьяснять вам не нужно). Для избавления от этого недостатка была придумана, так называемая, однородная система координат, где все действия над обьектами было очень удобно производить с помощью МАТРИЦ. Если хотели произвести сразу много действий, то просто перемножали матрицы нужных операций и результирующую матрицу, содержащую в себе все преобразования, уже применяли к вершинам всего обьекта. Удобно и быстро. И теперь все системы просчета 3D графики работают на матрицах.

Что представляет собой матрица? Это обычная числовая таблица из строк и столбцов. Вот как она выглядит:

```
|a11 a12 a13 a14|- матрица 4x4. Всего 16 элементов. Каждый элемент имеет
|a21 a22 a23 a24|  обозначение. Буквой обозначается элемент, а цифрами место
|a31 a32 a33 a34|  расположения в матрице. Например "a23" - это элемент,
|a41 a42 a43 a44|  стоящий на второй строке в третьем столбце.
```

Именно такие матрицы 4x4 и использует в своей работе Direct3D. Описываются они обычным массивом из 16 элементов, размер каждого элемента 4 байта (Real4). Соответственно, размер в памяти, который занимает одна матрица равен 64 байтам.

Существуют три основные матрицы, которые мы должны знать:

• World Matrix (мировая матрица)
• View Matrix (матрица вида)
• Matrix Projection (матрица проекции).

"Мировая матрица" отвечает за преобразование всей сцены (вращение вокруг осей, перемещение и т.д.). Именно всей, а не какого - либо отдельного обьекта на ней. "Матрица вида" отвечает за то, какая часть сцены будет видна на экране. "Матрица проекции" отвечает за то, как трехмерные

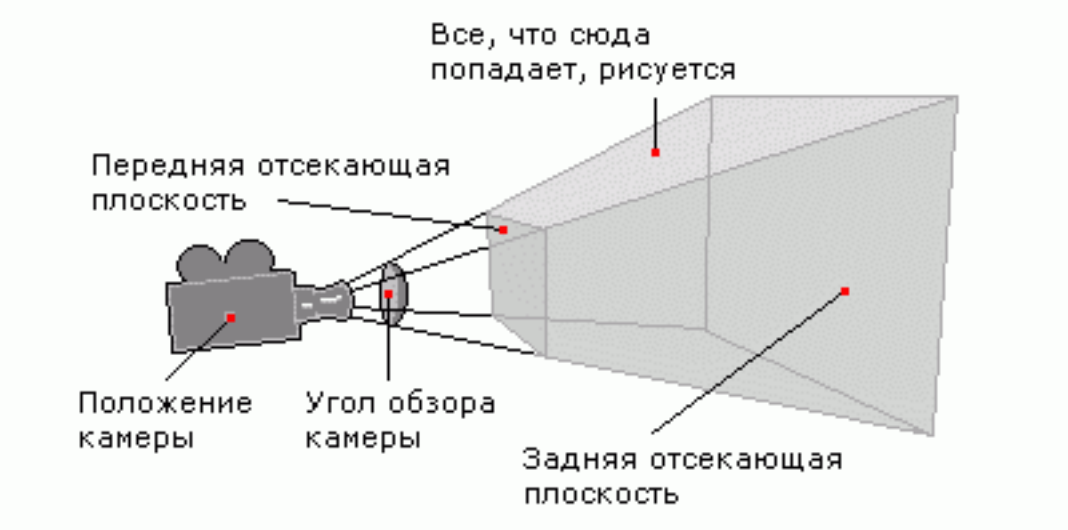
координаты будут преобразованы в двумерные для отображения на экран.

Эти три матрицы и являются тем чудесным средством, с помощью которого Direct3D превращает трехмерный мир в плоскую картинку на вашем мониторе.

5. Камера

Хоть трехмерный мир почти и бесконечен, но вот возможности ускорителя и процессора далеко не безразмерные. Если просчитать пару тысяч полигонов не проблема, то десятки и тем более сотни тысяч за раз подчас является трудоемкой задачей, даже для самых последних hi-end-овых ускорителей (естественно всякие там Cray не в счет :)). Поэтому опять приходится как-то оптимизировать все это дело. Решение простое. Чтобы все быстро просчитывалось и отрисовывалось с достаточно приемлемым числом кадров, наша сцена при просчете ограничивается в размерах. Осуществляется это намеренно. Сколько бы мы ни помещали на сцену обьектов, время просчета будет увеличиваться не так сильно, как при отсутствии такого ограничения. Вспомните про полигоны. Отбрасываются те, которые нам не нужны. Здесь отбрасываются те части сцены, которые не попадают внутрь этого ограничения.

В Direct3D роль такого ограничения выполняет камера. То что видно в камеру, то и отрисовывается.



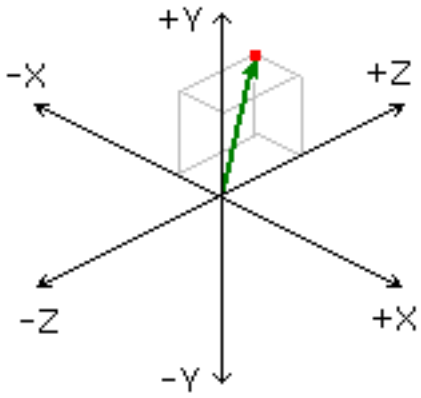
На рисунке видно, что ограничение представляет собой усеченную пирамиду, расположенную к нам вершиной. Помните про три чудесные матрицы? Так вот две из них и определяют камеру. Это "Матрица вида" и "Матрица проекции".

В "Матрице вида" мы задаем положение камеры в пространстве и ее направление. А заполнив нужными значениями "Матрицу проекции", мы тем самым определим какого размера будет эта ограничивающая пирамида.

Давайте остановимся на "Матрице вида" и разберем ее подробнее. Чтобы правильно заполнить эту матрицу нам необходимо задать значения трех векторов:

• EyeVector - координаты камеры в пространстве
• LookAtVector - указывает куда направлен обьектив камеры
• UpVector - указывает верх камеры

Если вы не знаете что такое вектор, то мы сейчас это исправим. Вектор - это направленный отрезок. Т.е. он указывает направление. Вот рисунок:



Вектор

Направление, которое описывает вектор, получается при взгляде от исходной в конечную точку. В нашем случае каждый вектор имеет только конечную точку. За начальную берется местоположение камеры в пространстве.

Задав эти три вектора, мы смело вызываем Direct3D и передаем ему все заботы, связанные с правильным заполнением "Матрицы вида" нужными значениями на их основе. Как всегда есть и альтернативный путь - заполнить матрицу самим, используя всякие преобразования над векторами (в документации по Direct3D имеется кое - какая информация), но пока не будем забивать голову и пойдем дальше.

"Матрицу вида" разобрали, осталась только "Матрица проекции". За нас эту матрицу тоже будет заполнять Direct3D. Вот какие значения мы должны ему передать:

• FieldOfView - поле обзора камеры
• AspectRatio - соотношение сторон
• NearViewPlan - передняя отсекающая плоскость
• FarViewPlan - задняя отсекающая плоскость

Поле обзора камеры - это угол в радианах, который способен охватить обьектив камеры (связано все это с фокусными расстояниями линз и пр.) Разный угол и по - разному будет выглядеть сцена. Как его правильно выбрать я расскажу далее.

Соотношение сторон - это то, в каком отношении находятся горизонталь и вертикаль. В обычном телевизоре соотношение сторон 4:3, т.е. если поделить 4 на 3 то получим 1.333333..... Вот это 1.33333 и нужно передавать Direct3D. Т.е. на вас лежит ответственность за его вычисление. Попробуйте вместо такого указать единицу и вы получите на экране с соотношением сторон 4:3 из квадрата прямоугольник, т.к. Direct3D будет пытаться соблюсти пропорции.

Передняя отсекающая плоскость - ближайшая к нам плоскость (вершина пирамиды). Задается значением по координате Z. Т.е. полигоны, находящиеся к нам ближе чем эта плоскость, не будут отрисованы.

Задняя отсекающая плоскость - тоже самое что и передняя, только наоборот (основание пирамиды).

В итоге задав три вектора и четыре параметра и вызвав Direct3D, мы получим готовые матрицы для использования. Камера готова !

6. Z буфер

Ну вот, какой-то там Z буфер еще надо знать. На самом деле это полезнейшая вещь. Если вы все еще не догадываетесь, то Z буфер позволяет Direct3D правильно отображать обьекты на экране.

Дело в том, что устройство визуализации рисует обьекты на экране в том порядке, в котором их передавать. Т.е. второй обьект нарисуется поверх первого, третий поверх второго и первого. Ну и что скажете вы? Если у нас статичная сцена, и мы передаем все обьекты в нужном порядке, то никакой Z буфер нам и впомине не нужен. Все правильно. А если сцена вращается, и еще обьекты меняют свое местоположение, что тогда? Тогда Z буфер незаменим.

Работает он так:

1. В начале отрисовки каждого кадра Z буфер заполняется нейтральным значением.
2. Когда просчитывается полигон, то его надо рисовать на экране. Он состоит из множества пикселей. У каждого из них есть координаты. Direct3D берет Z координату пикселя и сравнивает с тем значением, которое лежит в Z буфере. Если значение в Z буфере больше, то на его место заносится Z координата пикселя, а сам пиксель рисуется, если нет, то ничего делать не надо.
3. Берется следующий полигон. Выполняется пункт b). Так до тех пор пока все не будет отрисовано.

Говоря простым языком, проверяется на какой глубине в данном месте экрана нарисован пиксель, и если рисуемый пиксель находится к нам ближе чем тот, который уже нарисован, то старый затирается новым.

Z буфер имеет такие же размеры что и окно на экране. Если окно 320x240, то и Z буфер будет иметь размер 320x240. Размер значений, которые может содержать Z буфер, варьируются от 16 до 32 бит в зависимости от того, какой формат вы зададите. Соответственно, чем больше формат, тем больше времени будет уходить на его заполнение нейтральным значением. Такое заполнение называется очисткой Z буфера. При использовании буфера придется осуществлять такую очистку перед тем, как рисовать очередной кадр.

```
Справка

  Все возможные форматы Z буфера:
  -------------------------------
  1. D3DFMT_D16          - 16 битный.
  2. D3DFMT_D16_LOCKABLE - 16 битный. Блокируемый приложением.
  3. D3DFMT_D24X8        - 32 битный использующий только 24 бита.
  4. D3DFMT_D32          - 32 битный.
```

7. Все знания применим на деле !

Уфф... Ну как? Сложно? Думаю вы уже готовы все бросить и пойти отдохнуть. Действительно нужно время, чтобы переварить полученную информацию. Математика сложный предмет и с первого раза доходит не до всех. Я сам когда-то долго не мог въехать в некоторые термины :). Но когда приходится что-то изучать для любимого дела, то все понимается и усваивается гораздо быстрей.

Итак. Вы знаете что такое трехмерная система координат, вершина, полигон, матрица, вектор, сцена, камера и Z буфер. Теперь с этой информацией можно приступать к непосредственной работе с 3D.

В качестве примера я приготовил довольно занятную вещь. Обычные туториалы, которые встречались мне в инете, показывая работу простейшего трехмерного приложения, выводили на экран либо пару треугольников, либо вращающийся квадрат. Стопроцентно вы бы не хотели смотреть на этот примитивизм, да и я тоже. Из простых полигонов можно создать и нечто поинтереснее, не прилагая при этом слишком больших усилий. Вот наглядный пример:

То что вы видите, мы с вами и сделаем. Думаете сложно? Ничего подобного! Немного смекалки и умения. Если руки у вас растут из нужного места, то через пару уроков вы меня переплюнете и для вас не составит труда создать нечто похожее.

7.1 Об используемых библиотеках

При использовании в нашем приложении новых функций и методов, нужно подключать новые Inc файлы и библиотеки, тут мы останавливаемся перед выбором: либо подключать одну библиотеку, либо другую. Мне очень нравится, когда существует несколько путей для достижения желаемого. Microsoft'y видно тоже нравится, и специально для нас он приготовил две версии одной и той же библиотеки: d3dx8.lib и d3dx8d.lib. Разница у них не только в наличии буквы d, но и, как я понял, в реализации (может буква d символизирует о debug версии ?). Размер D3DX8D.LIB - 53 414 байт, и она содержит только ссылки на D3DX8D.DLL. А вот у D3DX8.LIB размер уже 2 с лишним метра, и она несет исходный текст функций d3dx внутри себя. Поэтому, когда мы пытаемся ее использовать для постройки приложения, текст всех функций, которые мы решили вызывать, включаются в наш exe'шник. К тому же эта библиотека еще ссылается на фунции, которые есть только в MSVCRT.DLL и в ADVAPI32.DLL. И чем больше мы используем разных функций, тем больше наш exe'шник. Исходя из всего выше изложенного выделим основные моменты:

```
При использовании D3DX8D.LIB
----------------------------
1. Нужна дополнительно D3DX8D.DLL ( размер около 700 кило )
2. В поставку DirectX 8.1 она не входила. И тот пакет, который у вас установлен
   на компе, чтобы играть в игрушки, ее не содержит.
3. Используя ее, вам придется распостранять свои творения вместе с этой DLL
4. Размер получаемого exe'шника такой, который вы сделали сами.

При использовании D3DX8.LIB
----------------------------
1. Никакой дополнительной DLL не требуется.
2. На стадии компиляции приложения необходимо подключать MSVCRT.lib
   и ADVAPI32.lib
3. Размер exe'ника вырастает пропорционально количеству используемых
   в нем функций, начиная от 150 килобайт и выше
   ( начальный размер зависит от версии Msvcrt.lib )
```

Да... Как ни крути, а все равно плохо. Выбирайте то, что вам по душе, а мне больше нравится вариант с DLL. Именно его я и буду использовать в дальнейшем. Чтобы не пришлось искать эту DLL по инету, я включил ее в архив с исходником. Поэтому при скачивании оного с Wasm.ru, не удивляйтесь большому размеру архива (сам exe примера занимает 7 кило).

7.2. Новое в диалоге

Для работы нам понадобится каркас от предыдущего урока. Как вы помните, он разбит на две части: диалог и само приложение. На форму диалога были добавлены 2 новых ComboBox, 2 простых текстовых метки и рамка. Из нового в диалоге - это возможность выбирать формат Z буфера и количество BackBuffer. BackBuffer определяется просто числом, поэтому из добавленного нами на форму ComboBox берем значение, которое выбрал пользователь, и помещаем в D3DPRESENT_PARAMETERS. А вот на определении формата Z буфера стоит остановится. Всего форматов Z буфера имеется 4 штуки. Нам придется проверять поддержку каждого из них и только затем помещать название этого формата в ComboBox. Сложность заключается в том, что при проверке нужно знать в каком разрешении будет запускатся приложение, а именно число бит на пиксель. Число бит Z буфера при создании устройства Direct3D не может быть больше, чем число бит включаемого разрешения. Например Z буфер имеет 24 бита, а включаемое разрешение 1024x768x16 бит. При таких параметрах нам будет отказано в создании устройства, т.к. 24 больше чем 16. Вот на это и стоит обратить внимание.

В самом файле Dialog.asm я объявил новые массивы: это ZbufferFormatTable - содержащий все возможные форматы Z буфера и ZbufferStrTable - содержащий указатели на текстовые строки с названиями этих самых форматов (да еще сами текстовые строки). Как и в предыдущем уроке проверяем, поддерживает данный формат ускоритель или нет, и если нужно заносим его название в ComboBox. Проверка осуществляется методом CheckDepthStencilMatch.

```
Справка

  CheckDepthStencilMatch передается 5 параметров.
  ------------------------------------------------
  1. Номер адаптера. D3DADAPTER_DEFAULT - адаптер по умолчанию.
  2. Тип адаптера. D3DDEVTYPE_HAL - аппаратный. D3DDEVTYPE_SW - программный.
  3. Формат поверхности адаптера ( какой на экране ).
  4. Формат поверхности BackBuffer.
  5. Формат Z буфера.
  ------------------------------------------------------
  Если все прошло успешно, метод возвращает D3D_OK.
  Если нет, то D3DERR_INVALIDCALL или D3DERR_NOTAVAILABLE.
```

Чтобы сообщить Direct3D что мы будем использовать Z буфер, нам еще придется заполнить дополнительно 2 параметра в D3DPRESENT_PARAMETERS. Это поле EnableAutoDepthStencil (заносим туда TRUE) и AutoDepthStencilFormat (заносим туда значение формата Z буфера, который пользователь выбрал в ComboBox). Все. Теперь наше приложение будет использовать Z буфер.

Все подробности смотрите в файле Dialog.asm

7.3. Новое в приложении

Чтобы мы могли использовать новые функции, нам необходимо подключить один Inc файл и одну библиотеку. Это d3dx8math.inc и d3dx8d.lib. Данные файлы поддерживают работу с матрицами и векторами, а также еще с несколькими математическими функциями на данный момент нам не нужными.

Изменились старые функции: Init_Direct3D, Destroy_Direct3D и Render_Scene.

Добавились новые: Set_Render_Parameters, Init_Scene и Animate_Scene.

7.4. Функция Init_Direct3D

Как обычно, после того как пользователь запустил приложение, мы получаем от диалога полностью заполненную структуру D3DPRESENT_PARAMETERS. На ее основе создаем устройство Direct3DDevice. Далее нам нужно создать камеру (о ней см. пункт 6).

```
Создаем камеру
--------------

.DATA
```

```
 EyeVector    D3DVECTOR  <0.0f, 0.0f, -2.6f>  ; Координаты камеры
 LookAtVector D3DVECTOR  <3.0f, -1.0f, 3.0f>  ; Куда смотрит камера
 UpVector     D3DVECTOR  <0.0f, 2.0f, -0.8f>  ; Указывает верх камеры

 FieldOfView    dd  0.7853981635f             ; Поле обзора камеры
 AspectRatio    dd  NULL                      ; AspectRatio
 NearViewPlanZ  dd  1.0f                      ; Передняя отсекающая плоскость
 FarViewPlanZ   dd  10.0f                     ; Задняя отсекающая плоскость

.DATA?

 WorldMatrix       D3DMATRIX  <?>             ; Мировая матрица
 ViewMatrix        D3DMATRIX  <?>             ; Матрица вида
 ProjectionMatrix  D3DMATRIX  <?>             ; Матрица проекции

.CODE

 fild    d3dpp.BackBufferWidth                ; Делим ширину на высоту
 fild    d3dpp.BackBufferHeight
 fdiv
 fstp    AspectRatio

 invoke  D3DXMatrixRotationY, ADDR WorldMatrix, NULL

 invoke  D3DXMatrixLookAtLH, ADDR ViewMatrix, ADDR EyeVector, \
         ADDR LookAtVector, ADDR UpVector

 invoke  D3DXMatrixPerspectiveFovLH, ADDR ProjectionMatrix, \
         FieldOfView, AspectRatio, NearViewPlanZ, FarViewPlanZ
```

Стандартно UpVector берется равным <0.0f, 1.0f, 0.0f> - т.е он направлен строго вверх. Наклоняя его в ту или иную сторону вы наклоняете камеру. Поле обзора камеры стандартно берется равным D3DX_PI/4. Т.е обычное число "Пи" делится на четыре. AspectRatio вычисляется делением ширины на высоту, и полученное число сохраняется.

Для заполнения WorldMatrix стандартно применяется функция D3DXMatrixRotationY с числом радиан равным 0. Получается, что мы хотим повернуть всю сцену на угол 0 по координате Y. В примерах Microsoft используется именно такой подход. Ради интереса я решил посмотреть, какими значениями заполняет эта функция пустую матрицу. Оказывается матрица из пустой просто превращается в единичную (единичная - это матрица из нулей у которой по главной диагонали стоят единицы), плюс один из элементов содержит какое-то число. Я попробовал сам заполнить матрицу как единичную, опуская то непонятное число. И что вы думаете? Все работает. По крайней мере для данного урока. Значит матрицы можно смело заполнять самим, не используя функции из библиотеки. Правда никто не гарантирует, что это будет легко (документация содержит информацию о том как формируются некоторые матрицы). Если вам интересно, пишите, и я рассмотрю эту тему.

ViewMatrix заполняем с помощью функции D3DXMatrixLookAtLH, передавая ей три вектора. А ProjectionMatrix заполняем функцией D3DXMatrixPerspectiveFovLH, передавая ей четыре параметра. Буквы LH в названии обеих функций говорят о том, что матрицы заполняются, исходя из левосторонней системы координат (см. пункт 1). Вызвав все эти функции, мы подготовили камеру к работе.

Осталось только заполнить сцену и установить параметры рендеринга. Сделаем это, вызвав написанные нами функции Init_Scene и Set_Render_Parameters. На этом Init_Direct3D заканчивается.

```
Справка

  D3DXMAtrixRotationY передается 2 параметра.
  -------------------------------------------
  1. Адрес матрицы в памяти.
  2. Угол в радианах.
  D3DXMAtrixLookAtLH передается 4 параметра.
```

```
----------------------------------------
1. Адрес матрицы в памяти.
2. Адрес EyeVector
3. Адрес LookAtVector
4. Адрес UpVector

D3DXMatrixPerspectiveFovLH передается 5 параметров.
------------------------------------------------
1. Адрес матрицы в памяти.
2. Поле обзора
3. AspectRatio
4. Значение переднего плана
5. Значение заднего плана
```

7.5. Функция Init_Scene

В этой функции я решил поместить подготовку сцены. На скриншоте вы видели некое подобие флага. Как его создать? Текстур мы не используем, все что у нас имеется - это треугольники, которые можно закрасить каким - либо цветом. Что делать? Давайте создадим флаг из квадратов и там, где нужно, закрасим квадраты красным цветом, чтобы получилась надпись.

Во - первых, описываем формат нашей вершины. Он будет содержать координаты и цвет.

```
.CONST
 D3DFVF_CUSTOMVERTEX equ D3DFVF_XYZ OR D3DFVF_DIFFUSE
```

Затем нам понадобится шаблон для клонирования:

```
.DATA
 Vertices  dd 0.0f , 0.0f , 0.0f , 0d1d0e7h   ; 4*4 байт на вершину
           dd 0.0f , 0.1f , 0.0f , 0d1d0e7h   ; 12*4 байт на полигон
           dd 0.1f , 0.1f , 0.0f , 0d1d0e7h   ; 24*4 байт на квадрат

           dd 0.0f , 0.0f , 0.0f , 0d1d0e7h
           dd 0.1f , 0.1f , 0.0f , 0d1d0e7h
           dd 0.1f , 0.0f , 0.0f , 0d1d0e7h
```

Здесь определено 2 треугольника, а все вместе - квадрат. Нужно размножить этот шаблон в количестве 36 по горизонтали и 10 по вертикали. Всего 360 квадратов или 720 треугольников.

Не забудем о надписи. Для ее нанесения используем обычную битовую маску для определения того, где закрашивать нужный квадрат. Надпись WASM.RU укладывается в шесть 32 битных значений. Вот они:

```
.DATA
 Text  dd 10001001100011001000100011001001b   ; Надпись WASM.RU
       dd 10001010010100001101100010101001b
       dd 10101010010011001010100010101001b
       dd 10101010010000101000100011001001b
       dd 10101011110100101000100010101001b
       dd 01010010010011001000101010100110b
```

Вопрос с флагом решен. Теперь подумаем, какой фон мы можем сделать. Обычный одноцветный можно получить, используя простую очистку экрана. Фон с градиентной заливкой выглядит привлекательнее. Возьмем еще два полигона, расположим их за флагом и раскрасим как нам нужно. Вот описание фона:

```
.DATA
 Fon  dd -1.0f, -6.0f, 1.5f, 0f1f0e7h    ; Фон - 2 полигона с цветом
      dd -1.0f,  2.0f, 1.5f, 0b1b0a7h    ; для градиентной заливки
      dd  9.0f,  2.0f, 1.5f, 0f1f0e7h

      dd -1.0f, -6.0f, 1.5f, 0f1f0e7h
      dd  9.0f,  2.0f, 1.5f, 0f1f0e7h
      dd  9.0f, -6.0f, 1.5f, 0FFFFFFh
```

С данными разобрались, и наступило время узнать как же Direct3D с ними работает. Система хранения данных в Direct3D основана на буферах. Существуют разные типы буферов. Например,

VertexBuffer - это буфер, содержащий данные вершин. Создав специальным методом нужный буфер, мы должны занести туда данные, и только затем указывать, что из него рисовать. После создания к буферу можно обращаться и менять какие - либо данные в нем. Переопределять размер в большую или меньшую сторону, как я знаю, нельзя (тема рассмотрена мной мало, возможно это как-то все-таки осуществляется). Вообще, ОЧЕНЬ не рекомендуется производить громоздкие манипуляции с буфером во время отрисовки сцены. Записывать данные еще можно, а вот считывать из него ... крайне не приветствуется. Создается VertexBuffer методом CreateVertexBuffer. Для нашего примера создание буфера выглядит так:

```
Создание Vertex буфера:
---------------------------------------
.CODE

d3dev8  CreateVertexBuffer, pd3dDev, (24*4)*360+24*4, D3DUSAGE_WRITEONLY, \
        D3DFVF_CUSTOMVERTEX , D3DPOOL_MANAGED, ADDR pd3dVertexBuffer

d3dvb8  Lock1, pd3dVertexBuffer, NULL, CountData, ADDR pLockBuffer, NULL
```

После создания буфера, доступ к нему осуществляется посредством блокирования. Оно необходимо, так как буфер может находится где угодно. В памяти на самом акселераторе, в AGP памяти, либо где-то еще. Методом Lock блокируем, а разблокирование осуществляется методом Unlock. Когда буфер блокирован, ускоритель не может с ним работать, учтите это хотя возможны и исключения.

Для нашего примера мы создаем буфер такого размера, чтобы в него как раз уместилось все необходимое. Затем блокируем его и начинаем заносить данные. Внимательно следите за размером заносимого, если он превысит размер буфера, то приложение вылетит с ошибкой о записи в недопустимое место памяти.

```
В первую очередь занесем данные фона:
-------------------------------------
lea     esi, Fon
mov     edi, pLockBuffer
mov     ecx, 24
rep     movsd

Следом клонируем данные флага посредством циклов:
-------------------------------------------------
.DATA
  xplus   dd 0.105f
  yplus   dd 0.0f
  shiftx  dd 0.105f
  shifty  dd -0.105f

.CODE
  push    edi
  mov     esi, edi
  push    10
  pop     ecx
  nextVertices3:
  xor     ebx, ebx
  push    ecx
  push    36
  pop     ecx
  nextVertices2:
  push    ecx
  lea     edi, Vertices
  push    6
  pop     ecx
  nextVertex:
  fld     DWORD PTR [edi]
  fadd    xplus
  fstp    DWORD PTR [esi]
  fld     DWORD PTR [edi+4]
  fadd    yplus
  fstp    DWORD PTR [esi+4]
```

```
push    DWORD PTR [edi+8]
pop     DWORD PTR [esi+8]
mov     eax, DWORD PTR [edi+12]
sub     eax, ebx
mov     DWORD PTR [esi+12], eax
add     esi, 16
add     edi, 16
loop    nextVertex
add     ebx, 00030303h
fld     xplus
fadd    shiftx
fstp    xplus
pop     ecx
loop    nextVertices2
push    shiftx
pop     xplus
fld     yplus
fadd    shifty
fstp    yplus
pop     ecx
dec     ecx
jnz     nextVertices3
```

Так как трехмерная графика основана на цифрах с плавающей запятой, то без сопроцессора не обойтись. При клонировании мы просто добавляем нужное число по какой - либо координате. У меня прибавление по X идет, начиная от 0 в плюсовую сторону. По Y от 0 в минусовую сторону.

В итоге флаг займет нижнюю правую четверть экрана. Почему так? Потому что начало трехмерной системы координат располагается по центру экрана. Если вы хотите иметь какое - то другое расположение системы координат, то вам придется или сдвигать камеру в нужную сторону, или заполнять "Мировую Матрицу" нужными данными. Я просто сдвинул камеру и все.

Мы не используем освещение, но можно его немного симмитировать. В цикле клонирования очередного квадрата по координате X заносится цвет более темного оттенка, чем у его предыдущего собрата.

```
Фон занесен, флаг тоже, осталась надпись:
----------------------------------------

pop     esi
add     esi, 96*74
lea     edi, Text
push    6
pop     ecx
nextVert2:
mov     ebx, 00FF0000h
push    ecx
mov     eax, [edi]
push    32
pop     ecx
nextVert:
shl     eax, 1
jnc     noMovColor
mov     DWORD PTR [esi+12], ebx
mov     DWORD PTR [esi+28], ebx
mov     DWORD PTR [esi+44], ebx
mov     DWORD PTR [esi+60], ebx
mov     DWORD PTR [esi+76], ebx
mov     DWORD PTR [esi+92], ebx
noMovColor:
add     esi, 96
sub     ebx, 00040000h
loop    nextVert
add     esi, 96*4
add     edi, 4
pop     ecx
loop    nextVert2

d3dvb8  Unlock, pd3dVertexBuffer       ; Разблокируем буфер
```

Посредством сдвига битовой маски определяем какой квадрат закрасить, одновременно увеличивая оттенок квадрата по координате X. Так как каждый квадрат состоит из двух треугольников, то всего надо заполнить цветом шесть вершин. Вообще Direct3D по умолчанию использует градиентную заливку, если у треугольника для каждой из вершин указать разный цвет, то он будет нарисован с плавными переходами из одного цвета в другой. Нам же нужен квадрат одного цвета, поэтому и приходится заносить одинаковый цвет во все шесть вершин.

Посмотрите на то, в какой последовательности я заношу данные в VertexBuffer. Сначала данные фона, а затем данные флага. Помните, в шестом пункте про Z буфер я упоминал, что все данные отрисовываются в порядке их следования? Фон находится дальше чем флаг, правильно? Значит нам по идее Z буфер вообще не нужен. Попробуйте его отключить, и посмотрите что произойдет. Все будет нарисовано как ни в чем не бывало. Используя такие вот особенности, можно увеличить скорость прорисовки, а именно FPS. Старайтесь отключать все, что только можно, и использовать лишь то, что действительно необходимо.

Все. Сцену можно рисовать в функции Render_Scene.

Хотелось бы еще остановится на одном важном моменте. Всегда записывайте на бумажку размер вашей вершины, чтобы не забыть. Исходя из него, нужно подсчитывать, сколько будет занимать один полигон, затем сколько будет занимать группа полигонов. Расчеты должны быть точными, если вы ошибетесь и, при создании буфера, его заполнении или при отрисовке, неправильно укажете данные, то рискуете ничего не увидеть на экране. Будьте очень внимательны.

```
Справка

  CreateVertexBuffer передается 5 параметров.
  ----------------------------------------
  Создает Vertex буфер необходимого размера

  1. Размер.
  2. Как будет использоваться. Значение является комбинацией флагов.
     D3DUSAGE_SOFTWAREPROCESSING - используется программная обработка вершин
     D3DUSAGE_DONOTCLIP          - содержимое буфера никогда не будет
                                   клиппироваться. Если установлен этот флаг
                                   то D3DRS_CLIPPING должен быть установлен
                                   в FALSE.
     D3DUSAGE_DYNAMIC            - буфер будет динамически использоваться.
                                   Если не указывать этот флаг, то буфер
                                   будет расположен в видео памяти.
                                   Если указать будет расположен в AGP памяти
                                   Флаг НЕ может использоваться совместно с
                                   D3DPOOL_MANAGED.
     D3DUSAGE_RTPATCHES          - буфер использует для отрисовки
                                   высокоорганизованные примитивы.
     D3DUSAGE_NPATCHES           - буфер использует для отрисовки N патчи
     D3DUSAGE_POINTS             - буфер использует для отрисовки точечные
                                   спрайты или список точек.
     D3DUSAGE_WRITEONLY          - только для записи. Direct3D расположит
                                   его по наиболее эффективному адресу.
  3. Формат вершины.
  4. Параметр определяющий как будет осуществляться менеджмент буфера.
     D3DPOOL_DEFAULT   - По умолчанию. После сброса устройства буфер
                         может быть потерян.
     D3DPOOL_MANAGED   - Direct3D будет управлять буфером.
                         После сброса устройства восстановление буфера не
                         требуется.
     D3DPOOL_SYSTEMMEM - расположить буфер в системной памяти. После сброса
                         устройства восстановление буфера не требуется.
     D3DPOOL_SCRATCH   - буфер расположен в системной памяти. После сброса
                         устройства восстановление буфера не требуется, т.к
                         драйвер о нем не знает. Можно производить с буфером
                         различные действия - копировать, блокировать и .т.д
                         ( не понимаю нафига этот флаг нужен? )
  5. Адрес ячейки куда будет помещен указатель на созданный буфер
```

```
-------------------------------------------------------
Если все прошло успешно, метод возвращает D3D_OK.
Если нет, то D3DERR_INVALIDCALL, D3DERR_OUTOFVIDEOMEMORY
или D3DERR_OUTOFMEMORY.

Lock передается 4 параметра.
----------------------------
1. Смещение в байтах относительно начала. Обычно используется 0.
2. Размер данных в байтах которые нужно блокировать. Если 0 то будет
   блокирован весь буфер.
3. Указатель на заполненный буфер.
4. Комбинация флагов. Как будет использоваться память при блокировании.
   D3DLOCK_DISCARD     - приложение отбрасывает этот буфер. Direct3D
                         возвращает указатель на буфер в новом месте и
                         производит рендеринг из старого места.
                         ( еще написано, что, вроде, скорость при этом не
                         падает, а наоборот может возрасти )
                         Этот флаг работает только, если буфер создан с
                         флагом D3DUSAGE_DYNAMIC.
   Насчет двух следующих флагов я не совсем понял, возможны ошибки.
   D3DLOCK_NOOVERWRITE - позволяет изменять содержимое буфера если он
                         уже был ранее блокирован но не был разблокирован
                         до нашего блокирования. Или вроде наоборот.
                         Можно включать для оптимизации когда приложение
                         только прибавляет данные в буфер.
                         Этот флаг работает только, если буфер создан с
                         флагом D3DUSAGE_DYNAMIC.
   D3DLOCK_NOSYSLOCK   - Позволяет системе исполнять свои обязанности
                         ( например движение курсора ) во время долгого
                         блокирования буфера. И т.д и т.п.
   D3DLOCK_READONLY    - Приложение не будет писать в буфер. Служит для
                         оптимизации. Не употребляется с D3DLOCK_DISCARD.
                         Нельзя использовать, если буфер создан с флагом
                         D3DUSAGE_WRITEONLY.
-----------------------------------------------
Если все прошло успешно, метод возвращает D3D_OK.
Если нет, то D3DERR_INVALIDCALL.

UnLock передается указатель на блокируемый буфер.
-------------------------------------------------
Если все прошло успешно, метод возвращает D3D_OK.
Если нет, то D3DERR_INVALIDCALL.
```

7.6. Функция Set_Render_Parameters

Все, что содержит данная функция, можно было поместить в Init_Direct3D. Почему этого не было сделано, вы сейчас поймете. В уроке номер 2 я рассказывал о проблеме потерянного устройства. При возвращении в наше приложение по Alt+Tab нам нужно сбрасывать устройство. При этом теряется ВСЯ информация, связанная с ним, и нам необходимо ее восстанавливать. В этой функции содержатся все настройки касательно параметров рендеринга. Мы можем легко их восстановить, поместив вызов этой функции сразу после сбрасывания устройства.

```
Установка параметров рендеринга:
--------------------------------

d3dev8  SetTransform, pd3dDevice, D3DTS_WORLD, ADDR WorldMatrix
d3dev8  SetTransform, pd3dDevice, D3DTS_VIEW, ADDR ViewMatrix
d3dev8  SetTransform, pd3dDevice, D3DTS_PROJECTION, ADDR ProjectionMatrix

d3dev8  SetRenderState, pd3dDevice, D3DRS_CULLMODE, D3DCULL_NONE
d3dev8  SetRenderState, pd3dDevice, D3DRS_LIGHTING, NULL
d3dev8  SetRenderState, pd3dDevice, D3DRS_ZENABLE, D3DZB_TRUE

d3dev8  SetVertexShader, pd3dDevice, D3DFVF_CUSTOMVERTEX
d3dev8  SetStreamSource, pd3dDevice, 0 , pd3dVertexBuffer, 16
```

Метод SetTransform позволяет устанавливать в механизме визуализации, все связанное с трансформацией. Вызвав его, мы окончательно даем знать Direct3D, что для просчета сцены будем

использовать те три матрицы, которые заполнили ранее.

Метод SetRenderState нужен для включения и выключения тех или иных настроек рендеринга. Он является практически незаменимым, так как с его помощью устанавливается большинство параметров. Использовать этот метод очень просто. При вызове указывается нужный нам обьект или свойство механизма визуализации и что с ним нужно сделать.

Так как в нашем примере нет обьемных тел, то мы можем отключить проверку следования вершин в полигоне: D3DRS_CULLMODE, D3DCULL_NONE. Также мы не используем освещение: D3DRS_LIGHTING, NULL. Но мы используем Z буфер: D3DRS_ZENABLE, D3DZB_TRUE.

Методом SetVertexShader мы указываем, что будет использоваться тот формат вершин, который мы определили сами. Методом SetStreamSource устанавливаем, что исходные данные для рендеринга нужно брать из буфера вершин, а число 16 означает размер нашей вершины в байтах.

```
Справка

  SetTransform передается 2 параметра.
  ------------------------------------
  1. Название обьекта, который будет модифицирован.
  2. Адрес расположения матрицы.
  ------------------------------------------------
  Если все прошло успешно, метод возвращает D3D_OK.
  Если нет, то D3DERR_INVALIDCALL.

  SetRenderState передается 2 параметра.
  --------------------------------------
  1. Название параметра, который будет модифицирован.
  2. Что с ним надо делать.
  ------------------------------------------------
  Если все прошло успешно, метод возвращает D3D_OK.
  Если нет, то D3DERR_INVALIDCALL.

  SetVertexShader передается указатель на созданный шейдер.
                  Или в простом случае просто указывается формат вершины.
  ------------------------------------------------
  Если все прошло успешно, метод возвращает D3D_OK.
  Если нет, то D3DERR_INVALIDCALL.

  SetStreamSource передается 3 параметра.
  ---------------------------------------
  1. Номер потока от 0 до максимального числа потоков -1
  2. Указатель на созданный VertexBuffer
  3. Размер элемента в буфере в байтах.
  ------------------------------------------------
  Если все прошло успешно, метод возвращает D3D_OK.
  Если нет, то D3DERR_INVALIDCALL.
```

7.7. Функция Render_Scene

Самая главная функция. В ней происходит вызов Direct3D, чтобы он отрендерил сцену, а затем показал на экран.

```
Полный текст функции:
---------------------

.DATA
 Zvalue      dd  1.0f
 clearcolor  dd  0

.CODE
 d3dev8  TestCooperativeLevel, pd3dDevice
 cmp     eax, D3DERR_DEVICELOST
 jne     noreset
 ret
 noreset:
 cmp     eax, D3DERR_DEVICENOTRESET
 jne     noreset2
```

```
    d3dev8  Reset, pd3dDevice, ADDR d3dpp.BackBufferWidth

    invoke  Set_Render_Parameters                          !!!!!!

    noreset2:

    invoke  Animate_Scene                          ; Анимация сцены

    d3dev8  Clear,pd3dDevice,0,NULL,D3DCLEAR_TARGET or D3DCLEAR_ZBUFFER,\
            0,Zvalue, 0

    d3dev8  DrawPrimitive, pd3dDevice, D3DPT_TRIANGLELIST, 0 , CountPrimitive
    d3dev8  Present, pd3dDevice, NULL, NULL, NULL, NULL

    ret
```

Самое главное, обратите внимание, где происходит вызов Set_Render_Parameters. Если его не произвести при возвращении по ALT+TAB, то приложение вылетит. Еще раз повторюсь, вы не должны забывать, что при сбросе устройства теряются практически ВСЕ установки, за редким исключением. Старайтесь определить все настройки в какое - либо одно место, чтобы при сбросе вам было легко их восстанавливать.

В предыдущих уроках мы использовали только очистку экрана цветом. Теперь с использованием Z буфера, нам необходимо очищать также и его. Соответственно в вызываемом нами методе Clear добавляем флаг D3DCLEAR_ZBUFFER и само значение для заполнения Zvalue. Стандартно Z буфер очищается значением 1.0f

Так как вся наша сцена состоит из однотипных треугольников, то ее можно отрисовать всего лишь одним методом DrawPrimitive. Этот метод служит для отрисовки определенного числа примитивов. В качестве параметров (для нашей сцены) ему нужно передать следующее: D3DPT_TRIANGLELIST - у нас одиночные треугольники, каждый из них задан тремя вершинами, начинать нужно с нулевого по счету, всего рисовать n-треугольников. Метод позволяет рисовать и выборочно какую-либо группу треугольников, достаточно указать с какого начинать + количество.

Ну и как обычно, все показывается на экран методом Present. Удивительно что на подготовку всей сцены и установку параметров уходит довольно много строк кода, а на отрисовку нужен всего лишь один метод. Для маленьких сцен это обычное дело, но для чего-то навороченного так легко будет не отделаться.

Да, чуть не забыл, перед тем как начинать отрисовывать сцену, всегда должен вызываться метод BeginScene, а в конце, когда уже все передали для просчета, метод EndScene. Т.е все команды для рисования должны находиться между этими двумя методами. Почему этого нет у меня? В документации написано, что если эти методы не будут вызваны, то Direct3D сам принудительно вызовет их. Не указано только, для каждого рисующего метода это будет сделано или нет. Так как у нас используется всего лишь один рисующий метод, то я решил выкинуть BeginScene и EndScene, ведь все равно Direct3D вызовет их за меня. В дальнейшем, когда я буду использовать много рисующих методов подряд, тогда и всякие BeginScene придется использовать, а пока все и так отлично работает. Если возможно сократить обьем вызываемого, почему не сделать этого, да и размер программы меньше :).

```
Справка

  DrawPrimitive передается 3 параметра.
  -------------------------------------
  Отрисовывает неиндексированные геометрические примитивы определяемого формата
  из установленного в StreamSource буфера

  1. Тип примитива.
     D3DPT_POINTLIST     - изолированные точки
     D3DPT_LINELIST      - изолированные линии. Если в буфере будет меньше чем
                           2 вершины, при вызове возникнет ошибка.
     D3DPT_LINESTRIP     - вершины образуют ломаную линию. Если в буфере будет
```

```
                        меньше чем 2 вершины, при вызове возникнет ошибка.
   D3DPT_TRIANGLELIST  - изолированные треугольники.
   D3DPT_TRIANGLESTRIP - сетка из треугольников.
   D3DPT_TRIANGLEFAN   - сетка из треугольников, но все они сходятся в
                         одной вершине.
2. Номер начального примитива
3. Количество примитивов
-----------------------------------------------
Если все прошло успешно, метод возвращает D3D_OK.
Если нет, то D3DERR_INVALIDCALL.
```

7.8. Функция Animate_Scene

Довольно сносный флаг у нас уже имеется. Но где вы видели абсолютно плоский флаг висящий в пространстве. Анимируем наш флаг, чтобы он смотрелся еще лучше. Обычная функция Sin или Cos поможет нам в этом. Конечно супер реалистичного эффекта мы не получим, но кое-что несомненно. Предлагаю вычислять значение Sin на определенной координате X и применять к координате Z. Получится, что квадраты, составляющие флаг, будут циклически плавать по Z.

Вот исходный код:

```
Анимация флага:
---------------

.DATA
  x                 dd 0.1f    ; Начальное значение
  x2                dd 0.0f
  z                 dd 0.0f
  zcoeff            dd 0.3f
  zcorrect          dd 10.0f
  sinx              dd 0.0f

.DATA?
  checktime         dd ?   ;
  color             db ?   ;
  colorshift        db ?   ;
  flag              dd ?   ;

.CODE
 invoke  GetTickCount      ; Получаем число тиков
 mov     ebx, checktime
 add     ebx, 50           ; Добавляем к нему коэффициент
 cmp     eax, ebx
 jb      noColor           ; Если не наступило время обновления, то выход
 mov     checktime, eax

 cmp     flag, 1           ; Здесь происходит проверка, которая позволяет
 je      goDec             ; узнать какой цвет был и какой нужно
 inc     color             ; использовать далее
 cmp     color, 164        ;
 jb      noxorFlag         ; Нужно для циклического изменения цвета фона
 xor     flag, 1           ; от синего до красного по кругу
 noxorFlag:                ;
 jmp     exitt             ;
 goDec:                    ;
 dec     color             ;
 cmp     color, 0          ;
 ja      exitt             ;
 xor     flag, 1           ;
 add     colorshift, 8     ;
 cmp     colorshift, 24    ;
 jb      noincshift        ;
 mov     colorshift, 0     ;
 noincshift:               ;
 exitt:                    ;

 d3dvb8  Lock1, pd3dVertexBuffer, NULL, CountData, ADDR pLockBuffer, NULL

 mov     esi, pLockBuffer            ; Обновляем цвет фона
 add     esi, 28                     ;
```

```
movzx   eax, color                  ;
movzx   ecx, colorshift             ;
shl     eax, cl                     ;
mov     [esi], eax                  ;

mov     edi, pLockBuffer            ; А здесь вычисляем нужное значение
add     edi, 104                    ; посредством Sin
fld     x2                          ;
fadd    x                           ;
fst     x2                          ;
fstp    z                           ;
push    36                          ;
pop     ecx                         ;
column:                             ;
push    ecx                         ;
fld     z                           ;
fsub    zcoeff                      ;
fst     z                           ;
fsin                                ;
fdiv    zcorrect                    ;
fstp    sinx                        ;
push    sinx                        ;
pop     eax                         ;
push    edi                         ; Обновляем Z координату в буфере
mov     ecx, 10                     ;
row:                                ;
mov     DWORD PTR [edi], eax        ;
mov     DWORD PTR [edi+16], eax     ;
mov     DWORD PTR [edi+32], eax     ;
mov     DWORD PTR [edi+48], eax     ;
mov     DWORD PTR [edi+64], eax     ;
mov     DWORD PTR [edi+80], eax     ;
add     edi, 96*36                  ;
loop    row                         ;
pop     edi                         ;
add     edi, 96                     ;
pop     ecx                         ;
loop    column                      ;

d3dvb8  Unlock, pd3dVertexBuffer
noColor:
ret
```

Чтобы была плавная анимация, нам необходимо обновлять значения, ориентируясь на время. Я использовал функцию GetTickCount. Можно использовать функцию timeGetTime из библиотеки winmm, говорят она будет поточнее. Обновление значений у меня происходит только 20 раз в секунду, этого вполне хватает. Как вы помните, при занесении данных в буфер нам нужно его блокировать, так вот здесь он блокируется также 20 раз в секунду. Все остальное время он не трогается. Приложение при этом рисует его с максимальным количеством кадров, которое возможно, т.е я никак не ограничиваю число кадров (при входе в функцию Animate_Scene просто проверяется наступило время обновления или нет. Если нет тогда сразу выход.). Также одновременно обновляется значение цвета у фона.

Итого, фон меняет окраску, а флаг развевается :) То, что мы здесь осуществили, является вертексной анимацией. Наверное слышали о такой? Как видите, совсем не сложно.

7.9. Функция Destroy_Direct3D

Уничтожаем все обьекты относящиеся к Direct3D.

```
d3dev8  Release, pd3dVertexBuffer  ; Уничтожаем Vertex буфер
d3dev8  Release, pd3dDevice        ; Уничтожаем Direct3DDevice8
d3d8    Release, pd3d              ; Уничтожаем Direct3D8
```

При уничтожении обьектов рекомендуется соблюдать порядок. Не думаю, что, если вы сразу уничтожите обьект Direct3D, то остальные будут уничтожены автоматически. Хотя возможно всякое. Кто его знает, что было у разработчиков на уме :).

8. Послесловие

Урок получился длинный. И хотя я старался расписать все как можно подробней, возможно некоторое сумбурное изложение с моей стороны. Для окончания позволю себе закрепить весь материал вкратце.

```
Порядок действий при запуске приложения:
-----------------------------------------
В диалоге уже создан обьект Direct3D
a) Создаем устройство Direct3DDevice
b) Заполняем три матрицы, вызывая Direct3D и передавая ему нужные параметры
c) Создаем VertexBuffer
d) Заносим в него данные, которые будем отрисовывать
e) Устанавливаем параметры рендеринга
   ( Устанавливаем три матрицы, включаем Z буфер, освещение и т.п. )

Все готово, можно отрисовывать сцену в цикле:
----------------------------------------------
f) Анимируем нужное
g) Очищаем BackBuffer и Z Буфер
h) Рисуем требуемое посредством DrawPrimitive
i) Показываем все на экран методом Present
j) Переходим на пункт F и т.д. пока не потребуется завершение программы

Завершение программы:
---------------------
k) Уничтожаем VertexBuffer
l) Уничтожаем обьект Direct3Ddevice
m) Уничтожаем обьект Direct3D
n) Завершаем приложение
```

PS:

В следующем уроке будет рассмотрена возможность использования обычного шрифта для вывода различной информации, ну и, может, в качестве примера подсчет FPS.

Как всегда исходник прилагается (находится в архиве `wasm_files.zip`: `files/0247/dx8masm04.zip`).

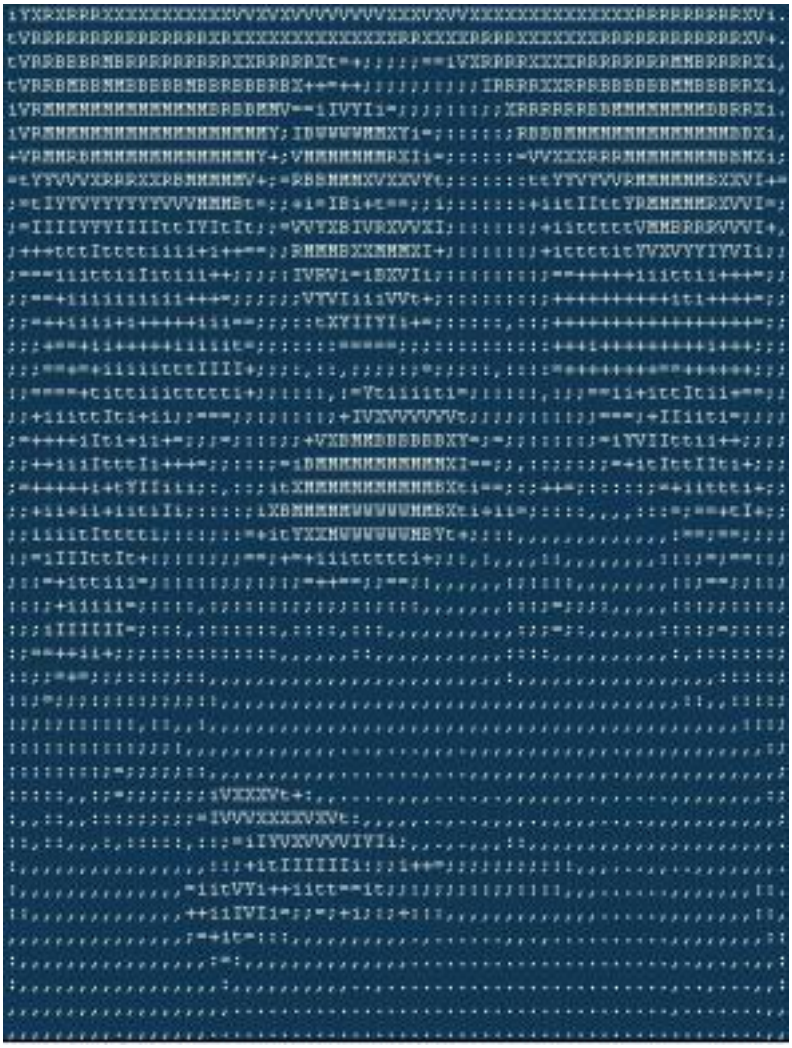
Все вопросы мыльте сюда mybox@aib.ru

MASM.OpenGL.0.Introduction

Автор: xzazet

Дата: 2003-07-08

Много тысяч лет тому назад наш прапрапрадедушка собрал травы и минералы и сделал из них краски. Рисовал он пальцами или использовал некое подобие кисти неизвестно. Но то, что создал этот первобытный Мастер до сих пор поражает воображение. Стремление человека выразить себя, рассказать о том, как прекрасен мир в котором он живет, до сих пор обитает во многих из нас. Времена изменились, появились новые технологии, но все еще не перестают удивлять нас звездное небо и краски восходящего солнца.



Почти сразу с первой вычислительной машиной появились настоящие произведения виртуального искусства. Картины рисовались на мониторах при помощи ASCII символов вручную. Уже потом появились программы, которые могли преобразовывать графическое изображение в ASCII-Art.

Появление мониторов с примитивным цветным графическим режимам, подняло виртуальное искусство на качественно новый уровень. Среди компьютерных фанатов появились художники, способные буквально из нескольких десятков байт создать нетривиальные анимированные графические изображения. У команд - "исследователей" коммерческих программ стали появляться визитные карточки - программы, рисовавшие оригинальные графические сцены. Они распространялись вместе с "патчами" (например, посмотрите файл protocols.com в архиве mental_klutt_by_psikorp.zip в конце этого текста).

Следующим этапом стало появление графических карт способных изображать достаточно убедительный 3D в реальном времени. Современная Demo Scene это виртуальное искусство, которое объединило в себе творчество художников и композиторов, людей с хорошей математической подготовкой, владеющих передовыми вычислительными технологиями.

Эти известные демо используют API OpenGL. Продолжайте чтение после просмотра h7-final.zip (находится в архиве `wasm_files.zip`: `files/0256/h7-final.zip`) и mentalkluttbypsikorp.zip (находится в архиве `wasm_files.zip`: `files/0256/mental_klutt_by_psikorp.zip`)

Как работать с этим текстом, чтобы хоть чему-нибудь научится

Этот текст предназначен для тех, кто впервые серьезно взялся за изучение **художественного 3D моделирования с помощью API OpenGL**. Прежде всего включите свое воображение, голову и поисковую систему www.google.com. Вам будут предложены задания в каждой главе, которые звучат примерно так "Постарайтесь понять в течении 10 минут как можно больше об использовании функции 'wglCreateContext'". Не пренебрегайте этим - знаний много никогда не бывает, а добывать их эффективно нужно учится. Без этого не стоит браться за вдумчивую работу с этим текстом.

Мы будем работать в MASM32v7 из пакета пропагандиста ассемблера под Windows - Hutch'a. Знание основ ассемблера - обязательно, а изучение Уроков Win32API от Iczelion - необходимо. Неплохо иметь представление об языке C/C++, иначе будет трудно понимать описание параметров функций API OpenGL. Мы постараемся научится творчески программировать, старайтесь внести в исходник урока что-нибудь свое, не бойтесь экспериментировать, изобретать свои "эффекты". Читатели с чрезмерной тягой к "copy-paste" подходу будут отчислены от занятий.

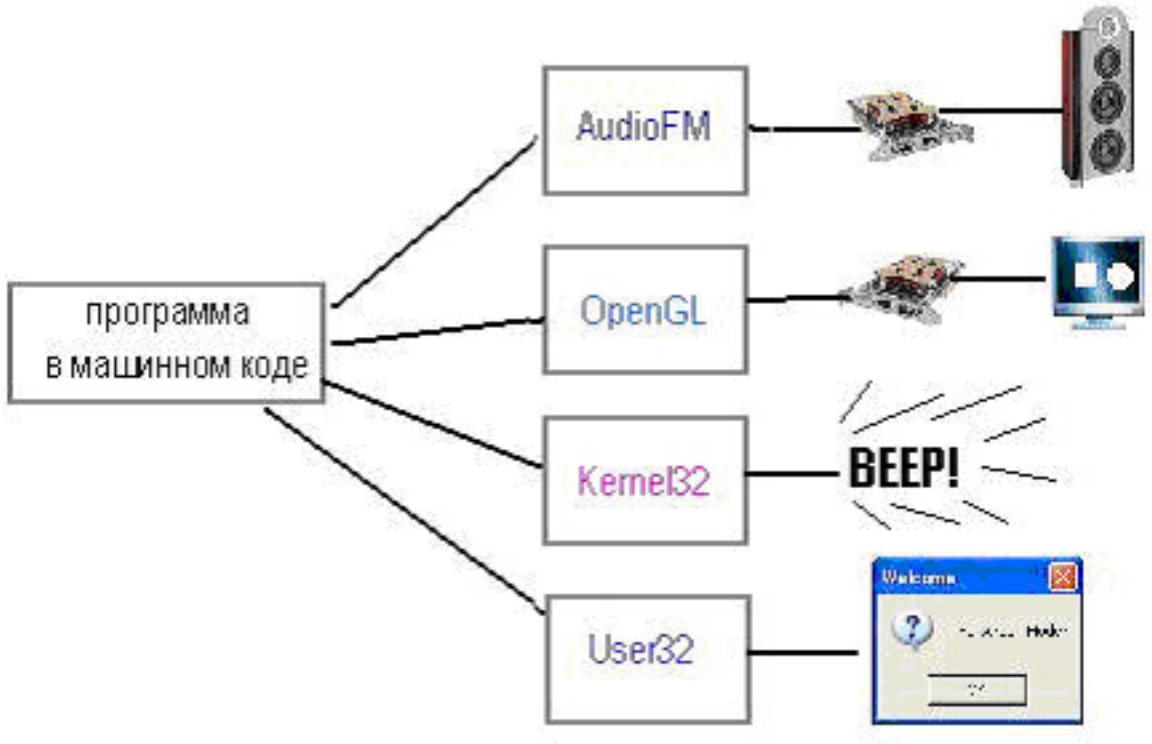
Лучше всегда иметь под рукой следующие мануалы:

1. Microsoft MASM v6.1 Assembly-Language Development System (Описание языка MASM)
2. MSDN Library Visual Studio (входит в пакет Visual Studio)
3. Сальвадор Дали "Дневник одного гения"
4. OpenGL Programming Guide aka "The Red Book"

Почему MASM и что такое API OpenGL?

Единственная причина выбора MASM - эстетическое удовольствие. Да, именно так. Кодировать на нем сложнее, он требует от Вас знаний намного более глубоких, чем язык Си. Хотя ассемблер и собирает программы меньшие по размеру (если сравнивать с VC) - не факт, что они будут всегда быстрее. Какой код (MASM или C/C++) будет выполнятся за меньшее время - зависит от Вас, от Вашего мастерства в нахождении самых быстрых алгоритмов - медленные алгоритмы на MASM с большой вероятностью проиграют в скорости хорошему коду на C.

Что же такое API OpenGL? Давайте договоримся, что под API (Application Programming Interface) мы будем понимать некий набор подпрограмм, которые сможем вызывать/исполнять из нашей программы. Тогда фраза "API OpenGL" будет звучать менее пугающе - это всего лишь набор подпрограмм для работы с 3D графикой. Теперь мы можем без труда понять, например, что означает фраза "Win32 API" - это просто набор подпрограмм для работы в операционной системе Windows. Рассмотрим следующий рисунок.



Художник очень упрощенно изобразил самую суть понятия API: его программа в машинном коде запускает/вызывает подпрограмму "MessageBox" из библиотеки "User32" (входит в состав Win32 API), которая показывает на экране небольшое диалоговое окно с кнопкой. Потом Художник заставил компьютер издать звуковой сигнал - просто запустил подпрограмму "Beep" из библиотеки "Kernel32" (также входит в Win32 API) Затем он обратился к API "AudioFM", которую написал его друг Музыкант. Художник разбирается в музыке плохо, а все-таки сумел воспроизвести вальс Шуберта (а затем и хруст французской булки). Как? Он просто вызвал подпрограмму из API "AudioFM".

Затем Художник мастерски изобразил на мониторе белый круг и белый же квадрат, используя API "OpenGL". Постойте-ка, а как он это сделал? Просто вызвал пару подпрограмм из библиотеки (=API) "OpenGL"!

Почему Художник написал на схеме "программа в машинном коде"? Чтобы особо подчеркнуть - и Asm и C/C++ сгенерируют машинный код, который будет работать с одним и тем же набором подпрограмм API "OpenGL". Так или иначе, но программы на ассемблере и программы C/C++ будут

обращаться к одним и тем же функциям API "OpenGL", а как быстро будет выполнятся код "OpenGL" зависит только от Вашей видеокарты. (Ассемблер или C++ - выбирать Вам, на скорость выполнения API OpenGL не влияет).

Все API (и API OpenGL в том числе) очень удобная штука. Попробуйте представить себе как работает ... например, телевизор. Что нужно сделать из транзисторов, ЭЛТ и электричества, чтобы телевизор включить, настроить каналы, подстроить цвета? Нам на спасенье к телевизору прилагается API "Пульт Дистанционного Управления". Нажал кнопку (вызвал подпрограмму) - включил телевизор, нажал другую кнопку - переключил на другой канал. Пульт работает с "железом" телевизора самостоятельно! А что будет если мы купим новую видеокарту? API OpenGL позаботится о "железе" (обратите внимание на связь API OpenGL и видеокарты на рисунке), а название ее подпрограмм и способ их вызова останутся прежними. Наша программа будет работать везде, где установлена API OpenGL и даже без помощи современных видеокарт (работать будет медленно, но работать будет :))

Итак, нам любезно предоставили дистанционный пульт управления (API "OpenGL") от наших видеокарт - на любом современном компьютере с операционной системой Windows уже установлена API OpenGL. А где у них та самая главная кнопка - "ВКЛ" мы увидим в следующей главе.

PS: Автор сознательно опустил некоторые, на его взгляд, несущественные нюансы - для облегчения понимания. Для любознательного читателя это не будет большой проблемой. В качестве разминки постарайтесь выяснить, верно ли это утверждение "API OpenGL доступен только на компьютерах с ОС Windows!", а также "Почему MASM не такой хороший язык для написания 3D редакторов на API OpenGL по сравнению, например, с Си".

MASM.OpenGL.1.First.Blood

Автор: xzazet

Дата: 2003-07-20

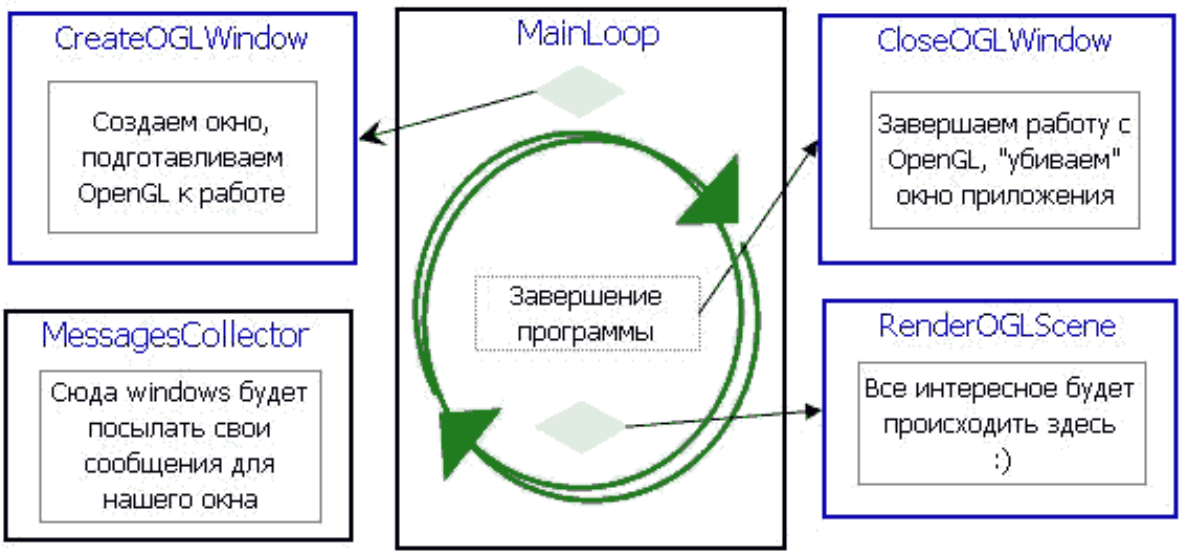
MASM.OPENGL - Глава 1
x z a z e t
787A617A6574406E61726F642E7275

MASM.OPENGL

GLьчатай, открой личико!

В предыдущей части мы занимались теорией, а сегодня пришло время практики. Давайте разбираться.

Скелет нашей первой программы



Синими рамками обведены процедуры нашей программы, в которых мы будем обращатся к подпрограммам API OpenGL. В принципе больше комментировать здесь нечего. Если вы хорошо поработали над уроками Win32API от Iczelion'a, остальная часть схемы (подпрограммы в черных рамках) не должна вызвать недоумения. Если это не так, то работу над Win32API придется продолжить. Остальным слушателям предлагаю перейти к следующей главе.

Таможня дает "добро"

Давайте посмотрим на подпрограмму `CreateOGLWindow` повнимательнее. Ее название говорит само за себя - мы создаем окно, в котором будут отображаться результаты нашего творчества в построении 3D виртуальной реальности. В следующей таблице расписаны шаги нашей подпрограммы, а также состояние OpenGL после каждого из них.

CreateOGLWindow	**Состояние OpenGL**
Создаем окно windows приложения	--неактивен--
Запрашиваем ОС Windows контекст устройства нашего окна	--неактивен--
Настраиваем контекст устройства нашего окна для работы с ним на OpenGL	--неактивен--
Сообщаем OpenGL о подготовленном контексте устройства	OpenGL создает контекст рендеринга (изображения), пригодный для работы с нашим окном (контекстом устройства)
Выбираем созданный OpenGL'ем контекст изображения (рендеринга) для построения 3D сцен командами API OpenGL	OpenGL готов к работе на выбранном контексте изображения - ожидает от нас команды для исполнения (путь в 3D мир открыт!)
MOV EAX, ECX (или любая другая)	OpenGL активен (ждет команды)

Сразу договоримся - мы будем смотреть на работу нашей программы с API OpenGL как на общение Клиента (наша программа) с Сервером (OpenGL). Представим на минутку, что мы пытаемся взломать секретный Сервер "OpenGL". Что нужно сделать сначала? Определить по какому сетевому протоколу нам удобнее приконектиться к серверу! Допустим, мы выбрали протокол ABC_32бита. Это хорошо, но настоящие "кулхацкеры" не вламываются на секретные сервера со своего домашнего компьютера! Они вламываются с убитых PCишек (напомню, мы работаем в MASM) деревенских интернет-кафе.

Поддерживается ли на них нужный нам протокол ABC_32бита? Нужно обязательно проверить его наличие перед использованием. Если он не поддерживается, то выбираем другой протокол или отказываемся от идеи взлома. По-моему, пока все логично. На этот раз нам повезло – PCшка поддерживает протокол ABC_16бит (хуже, чем ABC_32бита, но хоть что-то). Мы посылаем команду серверу OpenGL с запросом на соединение по протоколу ABC_16бит и получаем сообщение, что сервер готов к соединению. Дальше дело техники - сделать соединение активным. И всё, сервер OpenGL ожидает от нас команд для исполнения.

Не бойтесь контекстов устройств (device context) – они не кусаются. Вспомним наше детство: детский сад, строгую воспитательницу и неуемную страсть к рисованию. (на стенах, ватманах, на руках шариковой ручкой, на новеньком БМВ богатого дяди острым гвоздем). Но в детском саду у строгой воспитательницы всегда нужно было спрашивать ее разрешение перед тем, как расписывать что-нибудь – «Марья Ивановна (ОС Windows), можно я пририсую усы фломастером на вашей фотографии (в окне моей программы)?». А как иначе? За плохое поведение могут и из детского садика исключить («Ваша программа пыталась выполнить недопустимую операцию и будет закрыта» © WIndows), а там столько красивых игрушек!

Давайте договоримся, что под контекстом устройства (device context) мы будем понимать некий объект на котором нам не терпится что-либо нарисовать. В детском саду такими объектами (контекстами устройства) могли быть: лицо лучшего друга (зубной пастой во время «тихого часа»), стена в туалете, асфальтовая дорожка (цветными мелками – милое дело).

В мире Windows мы будем пользоваться 3-мя такими объектами (контекстами устройств) – монитором, принтером и памятью компьютера. API OpenGL может обрисовывать наши картины не

только на мониторе, но и на принтере (есть ограничения) и просто рисовать картинку в памяти компьютера (Off-screen rendering. Кстати, только в этом случае мы можем подправлять изображения OpenGL самостоятельно непосредственно в памяти.) Контекст изображения OpenGL (не путать с контекстом устройства) к Windows не имеет никакого отношения, это “внутренняя кухня” OGL.

Теперь, отягощенные знанием о контекстах устройств Windows и контексте изображения OpenGL, еще раз изучите таблицу «CreateOGLWindow».

Что автор имел в виду под «протоколом», например «ABC_32бита»?. Структуру под симпатичным названием PIXELFORMATDESCRIPTOR.

```
// C; © MSDN Library
typedef struct tagPIXELFORMATDESCRIPTOR { // pfd
    WORD  nSize;
    WORD  nVersion;
    DWORD dwFlags;
    BYTE  iPixelType;
    BYTE  cColorBits;
    BYTE  cRedBits;
    BYTE  cRedShift;
    BYTE  cGreenBits;
    BYTE  cGreenShift;
    BYTE  cBlueBits;
    BYTE  cBlueShift;
    BYTE  cAlphaBits;
    BYTE  cAlphaShift;
    BYTE  cAccumBits;
    BYTE  cAccumRedBits;
    BYTE  cAccumGreenBits;
    BYTE  cAccumBlueBits;
    BYTE  cAccumAlphaBits;
    BYTE  cDepthBits;
    BYTE  cStencilBits;
    BYTE  cAuxBuffers;
    BYTE  iLayerType;
    BYTE  bReserved;
    DWORD dwLayerMask;
    DWORD dwVisibleMask;
    DWORD dwDamageMask;
} PIXELFORMATDESCRIPTOR;
```

Я специально сделал фонт помельче, чтобы у вас не возникло желание с ним разбираться прямо сейчас. На данном этапе просто насладитесь количеством разных параметров, которые мы сможем выбирать для работы с OpenGL.

Для чего конкретно нужна эта структура? PIXELFORMATDESCRIPTOR определяет формат пикселей графического изображения, например, сколько цветов оно может отображать (16бит или 32бита - вы неоднократно делали этот выбор, играя в компьютерные игры). Механизм работы со структурой предельно простой. Мы «заказываем» нужный нам формат пикселей с которыми хотели бы работать в 3D. Затем отправляем «заказ» ОС Windows. Система подбирает нам наиболее подходящий формат из списка тех, что поддерживаются «железом» на данном компьютере и возвращает его порядковый номер (индекс). Мы сообщаем Windows, что будем работать в предложенном формате, и настраиваем контекст устройства окна под него. Теперь мы уверены, что контекст устройства окна и контекст изображения OpenGL будут работать в одинаковых форматах пикселей, т.е. не будет ситуации, когда OpenGL пытается запихнуть 32бита туда, где могут поместится только 16бит. Логично?

MASM.OpenGL.Chapters – первая кровь

Вот как выглядит подпрограмма CreateOGLWindow.

```
;
;  MASMv7
;
;-------------------------------------------------------
CreateOGLWindow proc
;-------------------------------------------------------
;  Резервируем место для индекса
;  формата пикселей, возвращаемого Windows
LOCAL PixelFormat :DWORD

; Регистрируем класс нашего окна
invoke RegisterClassEx, ADDR wc
; Ошибка? - "Ошибка при регистрации окна"
; и выходим

; Создаем окно win32 приложения
; Для окна с OpenGL необходимы WS_CLIPSIBLINGS и WS_CLIPCHILDREN !
; Выставляем размеры окна 400 x 400
invoke CreateWindowEx,
    NULL,
    ADDR szWinClass,
    ADDR szWinName,
    WS_CLIPSIBLINGS or WS_CLIPCHILDREN \
or WS_OVERLAPPEDWINDOW,
    0, 0,
    400,
    400,
    NULL,
    NULL,
    hInstance,
    NULL
; Ошибка? -  "Ошибка при создании окна"
; и выходим
mov hWnd, eax

; Подпрграмма GetDC  (входит в Win32API) возвращает
; нам указатель на контекст устройства (КУ) нашего окна
; "Марьиванновна, можно мне порисовать на вашей фотографии?"
; Марьиванновна (с кислым лицом) - "Да на, рисуй. У меня еще есть".
invoke GetDC, hWnd
; Ошибка - "Не могу получить указатель КУ окна"
; и выходим
mov hDC, eax

; Подпрграмма ChoosePixelFormat входит в Win32API.
; В структуре pfd :PIXELFORMATDESCRIPTOR
; лежит наш "заказ" на пиксельный формат (ПФ),
; который мы хотели бы использовать в OpenGL.
; Просим Windows подобрать нам подходящий
; ПФ из поддерживаемых железом на выбранном
; контексте устройства - hDC
invoke ChoosePixelFormat, hDC, ADDR pfd
; Ошибка - "Нет подходящего пиксельного формата."
; и выходим
mov PixelFormat, eax

; Подпрграмма SetPixelFormat входит в Win32API.
; Настраиваем наш контекст устройства (hDC) на
; работу с подходящим пиксельным форматом, также
; сообщаем некоторую дополнительную информацию
; из нашей структуры pfd :PIXELFORMATDESCRIPTOR
; (windows добавит к выбранному ПФ "тонкие настройки")
; "Соединение КУ с OpenGL возможно по протоколу ABC_XXbits"
invoke SetPixelFormat, hDC,PixelFormat, ADDR pfd
; Ошибка - "Установка ПФ не удалась."
; и выходим
```

```
; Подпрграмма wglCreateContext входит в API OpenGL (windows версия).
; Просим OpenGL создать у себя контекст изображения,
; который можно отобразить на контексте устройства нашего
; окна -- формат пикселей у КУ и КИ  одинаковый!
; "Подготовить OpenGL для коннекта по протоколу ABC_XXbits"
invoke wglCreateContext, hDC
; Ошибка - "Ошибка в создании контекста изображения."
; и выходим
mov hRC, eax

; Подпрграмма wglMakeCurrent входит в API OpenGL (windows версия).
; Делаем созданный котекст изображения (hRC) активным,
; сообщаем на каком контексте устройства (hDC) мы хотим видеть
; результаты выполнения наших комманд к OpenGL (сегодня мы рисуем
; на мониторе).  OpenGL готов к выполнению наших комманд, находится
; в режиме ожидания.
; Поздравляю, мы в OpenGLe - 3D открыт!
invoke wglMakeCurrent, hDC, hRC
; Ошибка - "Ошибка активизации КИ."
; и выходим

; Подпрграмма ShowWindow входит в Win32API.
; Являем OpenGL окно миру
invoke ShowWindow, hWnd, SW_SHOW

; Подпрграмма SetForegroundWindow входит в Win32API.
; Да не просто являем, а выпячиваем
invoke SetForegroundWindow, hWnd

; Подпрграмма SetFocus входит в Win32API.
; Да не просто являем и выпячиваем, а насильно
; приковываем к нему всеобщее внимание.
invoke SetFocus, hWnd

return TRUE
CreateGLWindow endp
```

Keep your country tidy! (англ.)

Как бы Вам понравилось, если на Вашей жилой площади совершенно чужие люди брали ваши кровные контекстные устройства и создавали разные контексты изображения, работали с ними в свое удовольствие, жутко при этом мусорили и, махнув ручкой на прощанье, убирались восвояси, не прибрав за собой? Вот и ОС Windows будет ужасно расстроена, если мы не приберемся.

Напомню, что, создавая наше первое OpenGL приложение, мы на время взяли для рисования контекст устройства окна. Затем по нашей просьбе добрый OpenGL выделил для программы частичку себя – контекст изображения. Настоящие джентльмены, поиграв в чужие игрушки, возвращают их хозяевам.

```
;
;  MASMv7
;
;--------------------------------------------------------
CloseOGLWindow proc
;--------------------------------------------------------

; Есть ли в нашем распоряжении контекст изображения OpenGL?
.IF hRC != 0

   ; Подпрограмма wglMakeCurrent входит в API OpenGL (windows версия).
   ; Сообщаем OpenGL, что больше не хотим активного соединения с его
   ; сервером. OpenGL перестает быть активным, все наши комманды ему
   ; будут проигнорированы, хотя созданный для  программы контекст
   ; изображения все еще зарезервирован.
   invoke wglMakeCurrent, NULL, NULL
```

```
    ; Подпрограмма wglDeleteContext входит в API OpenGL (windows версия).
    ; Мы сообщаем OpenGL, что выделенный для нас контекст изображения
    ; нам больше не понадобится. Сервер приводит себя в порядок, а мы идем
    ; своей дорогой. «И разошлись как в море корабли»
    invoke wglDeleteContext, hRC
.ENDIF

; Подпрограмма ReleaseDC входит в Win32API.
; Возвращаем фотографию Марьивановне с глубокой благодарностью
invoke ReleaseDC, hWnd, hDC

; Окно нашей программе больше не нужно
invoke DestroyWindow, hWnd

; Убираем за собой
invoke UnregisterClass, ADDR szClassName, hInstance

; Рисуем поразительно реальный 3D образ Клаудии Шифер, в полный рост и
; совершенно голую.
ret
; Шутка.
KillOGLWindow endp
```

PFD_SUPPORT_OPENGL equ 020h

Мы узнали почти все, чтобы закончить этот отрывок из MASM.OpenGL.Chapters. На сладкое осталась структура формата пикселей – pfd.

```
;
;  MASMv7
;
pfd PIXELFORMATDESCRIPTOR  { \
 sizeof PIXELFORMATDESCRIPTOR,
  1,
  PFD_DRAW_TO_WINDOW or  PFD_SUPPORT_OPENGL,
  PFD_TYPE_RGBA,
  32,
  0, 0, 0, 0, 0, 0,
  0,
  0,
  0,
  0, 0, 0, 0,
  0,
  0,
  0,
  0,
  0,
  0, 0, 0}
```

Нули нас пока не интересуют.

Единица это pfd.nVersion – версия структуры (да и такое бывает). nVersion всегда равна единице.

32 – это pfd.cColorBits, определяет кол-во битов на один пиксель (bbp), т.е. количество разнообразных цветов в которые можно этот пиксель покрасить. Я взял 32 (а мог бы 24 или 16, например), что даст мне возможность отобразить 4,294,967,295 цветов.

PFD_TYPE_RGBA (pfd. iPixelType)– сообщает Windows, что цвета пикселей мы будем определять через Red Green Blue компоненты (подробнее в следующих главах)

И последним определяем элемент pdf. dwFlags:

PFD_DRAW_TO_WINDOW сообщает Windows о том, что мы будет рендерить (строить 3D сцены) в окно, а не в память, например.

PFD_SUPPORT_OPENGL – у меня нет информации.

В этой главе мы казалось бы ничего не сделали особенного. То, что получилось, Вы увидите, когда после внимательнейшего изучения исходника; сами скомпилируете его, если надо – отловите ошибки, а затем запустите. Напоминаю, СНАЧАЛА изучаете исходник, ПОТОМ компилируете, выдерживаете одни сутки, и в самую последнюю очередь запускаете. Так и только так.

Один мой хороший знакомый добился исключительных результатов за непостижимо короткий срок. Когда я попытался выяснить, в чем секрет и какими пособиями он пользовался, ZYtDHE грустно посмотрел в мою сторону и виновато промямлил «Рассыпался винчестер, а на старом не было компилятора». (!!!) Этот парень создавал программы, отлавливал свои ошибки в течении двух недель, так не разу и не запустив код на исполнение. «All errors are made by mistake» (c) ZYtDHE.

Вы замечали, что первое впечатление от знакомства с чем-то новым - самое сильное? Если именно сейчас в эти первые минуты после прочтения текста вы постараетесь вникнуть в философию OpenGL, с самых азов, с вызова его первой подпрограммы, с инициализации его первой структуры, то остальной более продвинутый материал покажется вам логичным продолжением, вы увидите в хаосе нулей и единиц, байтов и двойных слов, в контекстах изображения удивительную красоту и гармонию. И тогда вы сможете импровизировать, как хороший музыкант на любимом инструменте, а возможности API OpenGL будут ограниченны лишь Вашей фантазией.

Информация к размышлению

Так как исходник без комментариев и обработки возможных ошибок после вызова подпрограмм API (на их месте стоят мои комментарии «и выходим», смотри выше по тексту), предлагаю следующий конкурс для любознательных – доработайте исходник. Лучший будет опубликован в следующей главе).

Еще необходимо предупредить читателя о том, что откомпилированный исходник может вообще не запуститься. Если это Ваш случай – остается только порадоваться. Ведь потратив некоторое время на самостоятельный поиск и исправление ошибок вы лучше усвоите материал, и значит потратили время не зря. Вам может понадобится отладчик. К сожалению, у меня сейчас при себе нет его номера телефона, а на email он не отвечает. Поищите его в Интернете. Встретите Сафонову Асю, передавайте привет.

Автор работает в редакторе AsmEditor, который можно скачать отсюда (http://www.avtlab.ru/) Так же скачайте и установите базу подсветки синтаксиса, которую любезно предоставил vkim, разработчик VKDEBUG (входит в пакет MASMv7).

Основные возможности AsmEditor:

• подключение различных компиляторов (Ассемблер, Си и другие);
• настраиваемые схемы подсветки ключевых слов;
• быстрый переход между процедурами, функциями, метками;
• возможность подключения файлов помощи (например, win32.hlp) для вызова контекстной справки по выделенному в редакторе слову;
• работа с исходными кодами в DOS-кодировке без потери символов псевдографики;
• меню и дополнительные кнопки с функциями, назначаемыми пользователем;
• настраиваемые "горячие клавиши"; быстрый переход по номеру строки; "закладки"; сохранение позиции курсора и т.д.

Не требует инсталляции, не вносит изменений в системные файлы и реестр. По умолчанию настроен на пакет MASM32

Также скачайте fraps (www.fraps.com) и испробуйте его на каких-нибудь программах.

Для развлечения, поиграйте с графическими режимами в своем любимом FPS, попереключайтесь с 16 на 32 цветность. Постарайтесь объяснить увиденное.

Просто вопросы на засыпку:

1. Если мы будем работать в режиме off-screen rendering'га (т.е. рисовать изображение в памяти компьютера, а не выводить на экран) сможем ли получить любой «заказанный» формат пикселей для роботы на этом контексте устройства (ограничений то по «железу» на память нет)?
2. Что такое wiggle?
3. Почему в нашей программе необходимо указывать WS_CLIPSIBLINGS и WS_CLIPCHILDREN при вызове CreateWindowEx?

DirectX 8.1 в MASM32: Урок 5. Использование текста

Автор: keYMax

Дата: 2003-08-01

Часто, когда приложение уже запущено, необходимо вывести некоторую полезную информацию: подсказки пользователю, содержимое переменных, сообщения об ошибках и т.д. Легко и быстро это делается с помощью текста.

Как вы все знаете, вывод текста в приложениях, не использующих 3D, осуществляется посредством GDI, который может предложить несколько функций, различающихся параметрами и удобством применения. В трехмерном же приложении использование GDI практически невозможно за редким исключением. Даже если удастся каким-то образом его задействовать, то потери времени при использовании будут очень значительные. В качестве альтернативы Direct3D предлагает нам всего лишь один метод для вывода текста. Если вы ожидаете от него каких-то наворотов и спецэффектов, то спешу вас разочаровать, метод служит только для простого вывода текста и ничего более. Кстати, у него имеется довольно много различных флагов, но они не сильно скрашивают и так хилые возможности. Более подробно на преимуществах и недостатках мы остановимся в конце статьи, а пока... пока рассмотрим с чего собственно начинать.

Первым делом, необходимо подключить .inc файл, содержащий прототипы необходимых нам функций и методов. Называется он d3dx8core.

Далее у нас есть два пути создания шрифта. Почему два и не более? Просто разработчики создали по аналогии с GDI две функции: D3DXCreateFont и D3DXCreateFontIndirect, которые и помогут сделать всю работу.

Рассмотрим по шагам первый путь:

```
1. а) Или вызываем функцию GDI CreateFont для создания шрифта
   б) Или заполняем структуру LOGFONT нужными параметрами и вызываем
      CreateFontIndirect

      ( те, кто не пользовался этими функциями, могут прочитать о них в одном
        из уроков Iczeliona )

2. Вызываем функцию D3DXCreateFont, передавая ей полученный ID шрифта
```

Вот и все. Проще просто не бывает.

А теперь второй путь:

```
1. Заполняем структуру LOGFONT нужными параметрами
2. Вызываем функцию D3DXCreateFontIndirect, передавая ей указатель на
   структуру
```

Второй путь самый минимальный. Никаких тебе GDI функций и лишней головной боли.

Справка

```
D3DXCreateFont передается 3 параметра.
-------------------------------------------
1. Указатель на Direct3DDevice
2. ID шрифта
3. Адрес памяти куда Direct3D поместит указатель на шрифт

Если все прошло успешно, функция возвращает D3D_OK.
Если нет, то D3DERR_INVALIDCALL или D3DERR_OUTOFMEMORY.

D3DXCreateFontIndirect передается 3 параметра.
-----------------------------------------------
1. Указатель на Direct3DDevice
2. Адрес структуры LOGFONT
3. Адрес памяти куда Direct3D поместит указатель на шрифт

Если все прошло успешно, функция возвращает D3D_OK.
Если нет, то D3DERR_INVALIDCALL или D3DERR_OUTOFMEMORY.
```

Как видно из справки, без уже созданного устройства Direct3DDevice создать и зарегистрировать шрифт не представляется возможным.

После того, как мы получили от Direct3D указатель на шрифт, можно начинать с ним работать. Как уже упоминалось выше, у нас имеется в распоряжении только один метод отрисовки текста, он называется DrawText и существует в двух вариантах: DrawTextA - рисует текст в формате ANSI, DrawTextW - рисует текст в формате UNICODE.

Отрисовка не должна вызвать у вас затруднений, но на всякий случай я приведу пример:

```
Почистили буфер...
------------------
d3dev8  Clear, pd3dDevice, 0, NULL, D3DCLEAR_TARGET or D3DCLEAR_ZBUFFER, \
               clearcolor , Zvalue , 0

Отрисовали полигоны...
----------------------
d3dev8  DrawPrimitive, pd3dDevice, D3DPT_TRIANGLELIST, 0, CountPrimitive

А теперь рисуем текст...
------------------------
d3dxfont8 DrawTextA, pd3dFont, ADDR szText, -1, ADDR RECT, DT_NOCLIP, colortext

И все показываем...
-------------------
d3dev8  Present, pd3dDevice, NULL, NULL, NULL, NULL
```

Отрисовку текста нужно производить самым последним этапом перед показом буфера на экран. Думаю это обьяснять вам не надо.

Справка

```
DrawText(A или W) передается 5 параметров.
------------------------------------------------------------------------
1. Адрес текста который нужно вывести
2. Число символов в строке или -1, чтобы Direct3D сам вычислил его
3. Адрес структуры типа RECT
4. Сумма флагов
5. Цвет текста в формате ARGB

Флаги: ( приведены только самые используемые, специфические опущены )
------------------------------------------------------------------------
DT_BOTTOM      - выравнивает текст по нижней границе прямоугольника
                 Может использоваться совместно с DT_SINGLELINE

DT_LEFT        - выравнивает текст по левой границе прямоугольника

DT_RIGHT       - выравнивает текст по правой границе прямоугольника

DT_TOP         - выравнивает текст по верхней границе прямоугольника

DT_CENTER      - центрирует текст по горизонтали

DT_VCENTER     - центрирует текст по вертикали

DT_CALCRECT    - Direct3D сам вычислит прямоугольник, в который помещается
                 выводимый текст. Важно помнить, что при этом текст
                 отрисован не будет.

DT_NOCLIP      - не проверять выходит ли текст за границы прямоугольника

Внимание !
- Все флаги выравнивания текста действуют для того прямоугольника, в котором
  он должен быть выведен.
- Метод действует только с теми шрифтами, в которых Escapement и Orientation
  равны нулю.
- Если размер шрифта больше, чем область в которую он должен быть выведен,
  то ничего выведено не будет.

Если все прошло успешно, метод возвращает высоту текста в логич. единицах
Если нет, то - ноль
```

Ну вот, почти всю информацию относительно DrawText вы получили. Осталось только упомянуть о нескольких важных моментах.

В документации сказано, что вызов DrawText обязательно должен осуществляться внутри методов BeginScene и EndScene (они относятся к Direct3DDevice). Если проигнорировать это утверждение, то якобы нас ожидает кара небесная. Использовав DrawText без этих методов ничего страшного я не обнаружил. Также у самого D3DXFont имеются методы Begin и End. Они тоже призваны служить обрамляющими DrawText. Т.е вызов отрисовки текста должен проходит опять таки между ними. Но и без этих методов все прекрасно работает. Далее решайте сами использовать их или нет.

Второй важный момент заключается в использовании флага DT_NOCLIP. Используйте его всегда! Лучше заранее подсчитать входит текст в прямоугольник или нет, чем ждать от Direct3D что он обрежет слова. Использование этого флага РЕАЛЬНО повышает скорость прорисовки.

Еще очень важно НЕ использовать шрифты больших размеров, чем больше размер тем дольше все отрисовывается. У меня с увеличением размера букв происходило падение fps и довольно заметно. Ну и соответственно это количество выводимого, если его много, тогда опять все тормозней и т.д. и т.п.

Стоит упомянуть и про потерю устройства, которую мы рассматривали в одном из предыдущих уроков. При сбрасывании устройства нам нужно восстанавливать все данные и о шрифтах. Делается это двумя методами OnLostDevice и OnResetDevice. Вызывать их нужно при каждой

потере устройства.

Справка

```
OnLostDevice и OnResetDevice ничего не передается.
При удачном завершении возвращается 0. При ошибке D3DERR_INVALIDCALL

Примечание
----------
В системе Windows 2000 порядок вызова данных методов должен быть следующим:

1. d3dxfont8  OnLostDevice, pd3dFont
2. d3dev8     Reset, pd3dDevice, ADDR d3dpp.BackBufferWidth
3. d3dxfont8  OnResetDevice, pd3dFont

Системы 98 и XP более лояльны и вполне переваривают такой порядок:

1. d3dev8     Reset, pd3dDevice, ADDR d3dpp.BackBufferWidth
2. d3dxfont8  OnLostDevice, pd3dFont
3. d3dxfont8  OnResetDevice, pd3dFont

Если вы будете использовать несколько шрифтов в свое программе, то при
Потере устройства достаточно вызвать OnLostDevice и OnResetDevice только
для самого первого. Все остальные шрифты будут восстановлены автоматически.
```

Теперь о цвете шрифта. Он используется в формате ARGB, где буква A - альфа или прозрачность. Чем больше альфа тем непрозрачнее текст. Когда я впервые пытался отрисовать текст и ничего на экране не увидел, то не сразу сообразил в чем дело (альфа была в ноль и соответственно текст абсолютно прозрачен). Не повторяйте мою ошибку, всегда проверяйте значение альфа.

Ну и на последок, как я и обещал выше, поговорим о преимуществах и недостатках.

Преимущества:

С помощью данного метода можно легко и быстро вывести информацию с подчеркиванием нужных букв, выравниванием, а также многострочный текст одним махом. Даже хоть справа налево по - арабски.

Недостатки:

Не является слишком быстрым и расторопным. Нельзя напрямую указать координаты вывода, только посредством структуры типа RECT (приходится ее все время модифицировать для вывода текста в различных местах).

Вывод:

Подходит только для простого отображения не слишком большого количества текста.

В качестве примера даже придумывать нечего, я попробовал его хоть как-то поинтересней сделать. Что из этого получилось, судить вам. Скачивайте и смотрите непосредственно в действии. На этом спешу с вами попрощаться.

Да ! Чуть было не забыл, теперь стабильность работы всех примеров, начиная с этого урока проверяется в системах Windows 98, 2000 SP3, XP и под DirectX 8.1 и 9.0. Надеюсь, никаких проблем у вас при запуске примеров возникнуть не должно, как было ранее в Win2k.

PS:

В следующий раз будет рассмотрено использование стандартных обьектов типа куб, сфера, цилиндр и т.д.

Как всегда исходник прилагается (находится в архиве `wasm_files.zip`: `files/0264/dx8masm05.zip`).

Все вопросы мыльте сюда mybox@aib.ru

FASM OpenGL tutorial

Автор: Tyler Durden

Дата: 2005-03-26

Глава 1: Бредисловие (вместо педисловия)

Привет всем (уважаемым читателям/телезрителям/"а также их родителям" и пр ;) Сразу хочу предупредить, что это моя первая статья, поэтому просьба не пинать меня ногами, никакого заме-е-ечательного Набоковского слога и живописных эпитетов вы тут не увидите :(Все выдержано в сухом "преподавательском" тоне без малейшего намека на "программерские" шутки про "юзеров", телепузиков, "Microsoft" и "г-на Била Гейтса" (а ровно и пауз на неопределенный срок фразой "я ушел бухать", так как я вовсе не "бухаю". Да да, не смейтесь, я спротЦмен ;)) Разочаровавшись (многие уже потянулись к "Esc"/бутылке/соске) начнем собсен-н-но...

Глава 2: Немного истории

Как говорил товарищ Берлага, если приподнять завесу, то становится весьма очевидным что стандарт OpenGL разрабатывается фирмой Silicon Graphics (теперь SGI) с лета 1992 года. Этот программный интерфейс был во многом основан на библиотеке IRIS GL, поддерживаемой с 1982 года. Для развития нового интерфейса был организован консорциум The OpenGL Architecture Review Board (ARB). На данный момент OpenGL является стандартной библиотекой для всех 32 разрядных операционных систем (Unix, Mac, Windows). Ну и хватит на этом. В конце концов это вам не лезгинка, а твист... в смысле не "Большая Советская Энциклопедия".

Глава 3: Инструментарий

Сразу скажу что писать мы будем на ассемблере (вообще говоря, все следует писать на ассемблере ;) , а точнее на FlatAssembler (FASM) т.к. он очень быстрый, компактный, постоянно развивается и не отягощен всякими бредовыми конструкциями (типа assume/offset) других ассемблеров (типа TASM ;)... а, ну да, самое главное, для читателей откуда-нибудь из Изгггаеля, его еще и на шару (конечно же под небо-о-ольшой процЭнт ;) можно скачать на www.flatassembler.net .Берем оттуда версию "for Win32 GUI" которая еще и, до кучи, оснащена простеньким IDE ! Далее... Распаковываем добро из скаченного архива "fasmw.zip" (находится в архиве `wasm_files.zip`: `files/0366/OpenGLbyTylerDurden.zip`) в какую-нибудь папку, и "усе готово шэф" !

Глава 4: Первое приложение (Simple)

Я предполагаю что вы уже писали под Win32 хотя бы и "Hello world !" (если нет то почитайте какой-нибудь tutorial по этому поводу, благо, их в сети туева хуча расплодилась). Окинув взглядом прилагаемый "simple.asm" в папке "simple" вы заметили что в нем нету даже оконной процедуры... Ой фу-у-у-у ! Но не спешите закрывать fasmw.exe, это всего лишь следствие моих небольших "оптимизаций" (а иначе накой хрен нам писать на ассемблере, как не из извращенческих соображений оптимизации ?) Но обо всем по порядку... Надеюсь следующий код понятен всем ? (fasminc можно установить например так (в файле autoexec.nt для WinXP) "set fasminc=путь_к_папке_fasm")

```
;формат exe файла
format PE GUI 4.0
;точка входа программы
entry start
;включаем файлы API процедур
;(должна быть установлена переменная окружения "fasminc")
include '%fasminc%\win32a.inc'
;констант OpenGL
include '..\include\opengl_const.inc'
;и макросов
include '..\include\opengl_macros.inc'
;начало программы
start:
```

Для начала спрячем курсор, дабы некоторые "продвинутые" юзвери не начали тыкать им куда не попадя, пораниться ведь можно (Извращения начинаются, см. "push ebx")

```
;обнулим ebx. Т.к. он не изменяется API процедурами
;то будем использовать push ebx вместо push 0, для оптимизации
xor ebx,ebx
;спрячим курсор
invoke ShowCursor,ebx
```

Для начала "урока рисования" нам нужно создать окно, а точнее его класс (и тут не без извращений ;)) размером с экран. Можно, конечно, для начала изменить разрешение, а потом уже уверенно создавать окно с известным размером, но это будет слишком просто для нас (и долго) ;) Поэтому мы получим текущее разрешение экрана по X и Y (по Z пока не будем ;)) и запихнем их в стек (перед этим забьем его нулями (ebx) это у нас будут другие параметры процедуры CreateWindowEx. Вообще многие функции подставляют вместо "0" (ebx) параметры "по умолчанию", этим, как говорит Microsoft, "интеллектуальным свойством" мы будем активно пользоваться во всех частях tutorial'a) Помним при этом что функции возвращают результат в "eax". Вообще, следите за стеком ! Это хоть и геморойно, но зато позволяет не сохранять кучу ненужных "промежуточных" результатов функций (вроде GetSystemMetrics).

```
;поместим в стек 4-е "0" для процедуры "CreateWindowEx"
push ebx
push ebx
push ebx
push ebx
;получим текущее разрешение по вертикали
invoke GetSystemMetrics,SM_CYSCREEN
;поместим его в стек для процедуры "CreateWindowEx"
push eax
;и по горизонтали
invoke GetSystemMetrics,ebx
;и его в стек
push eax
```

Тут мы используем полученное разрешение экрана (его берем уже из стека... а накой нам лишние переменные ?) чтобы найти его (экрана) AspectRatio (разрешающую способность, т.е. "max_x/max_y") Потом она нам еще пригодится

```
;вычислим соотношение разрешений экрана по горизонтали и вертикали
fild dword [esp]
fidiv dword [esp+4]
;и сохраним его в ratio
fstp [ratio]
```

По идее, для создания класса окна нам его нужно было бы с начала описать, зарегистрировать... В общем, скука ужасная (и еще байтов жрет кучу !) Вместо этого мы просто создадим окно с предопределенным классом, таких в винде множество, но мы выберем ну хотя бы класс "edit" (о чем красноречиво сообщает строка szClass) Помним о том что некоторые "недостающие" в invoke'е параметры мы уже положили в стек

```
;создадим окно размером с экран с предопределенным классом "edit"
;(т.к. его регистрировать не надо,
;то это позволяет избавиться от не нужного кода)
invoke CreateWindowEx,WS_EX_TOPMOST,\
  szClass,szTitle,WS_VISIBLE+WS_POPUP,ebx,ebx
```

И так, окно мы создали. Если вы уже рисовали что-нибудь под Win32, то вы знаете что для этого нужно получить его "контекст", это делается довольно просто (так как контекст окна нам еще пригодится, мы сохраним его в одном из "неиспользуемых" (в функциях Win32 API) регистров (ebx,ebp,esi,edi))

```
;получим контекст окна
invoke GetDC,eax
;сохраним его в ebp
xchg ebp,eax
```

Теперь нужно задать режим работы OpenGl'а, число бит на пиксель, режимы отображение и прочую белиберду (половину из которой мы оставляем нулевой (т.е. по умолчанию)

```
;инициализируем дескриптор формата
;пикселей OpenGL (поддержку OpenGL и двойной буферизации)
mov [pfd.dwFlags],PFD_DRAW_TO_WINDOW+\
 PFD_SUPPORT_OPENGL+PFD_DOUBLEBUFFER
;тип пикселей RedGreenBlueAlpha
mov [pfd.iPixelType],PFD_TYPE_RGBA
;глубину цвета
mov [pfd.cColorBits],32
;плоскость отображения
mov [pfd.dwLayerMask],PFD_MAIN_PLANE
;выберем его
invoke ChoosePixelFormat,ebp,pfd
;и установим его
invoke SetPixelFormat,ebp,eax,pfd
```

У нас есть контекст окна, у нас есть пиксельный формат ! "Мы сделаем из тебя парень новую звезду"... Аннннн нет, для начала нам еще осталось преобразовать контекст окна в "нечто" понятное OpenGL'у (т.к OpenGL не должен знать что такое Windows, для этого используется специальная "windows-compliant" функция API wgl*) и еще сделать его "текущим" (т.е. рисовать будем в него)

```
;преобразуем контекст окна в контекст OpenGL
invoke wglCreateContext,ebp
;и сделаем его текущим
invoke wglMakeCurrent,ebp,eax
```

Фууу-х-х-х-х ! Самое сложное позади ;) Винды остались не у дел, теперь "познакомимся", собственно, с простым и удобным OpenGL'ем ;) Говоря голосом ведущего "в мире животных", эта забавная зверушка тоже требует некоторой настройки параметров (ОПЯТЬ ?!) Дело в том что нам нужно включить "перспективные преобразования", проще говоря перспективу, она зависит от текущей разрешающей способности экрана (т.е. max_x/max_y) вот для этого то мы ее и вычисляли. Так как некоторые извращенческие функции OpenGL воспринимают только параметры типа Double (8 байт) то придется пихать их в стек макросом glCall (который использует макрос glPush) А шо

делать ? У нас же Intel Все-таки :(Число 90.0 это требуемый "угол обзора" можно поставить его 45.0 или 60.0 как кому нравится. Числа 0.1 и 100.0 это координаты "отсекающих плоскостей", все, что находится между ними, будет отображено на экране. Остальное - обрезано. Надо заметить что многие функции API OpenGL (glRotate, glTransform и пр.) работают с "текущей матрицей", коих 3 (GL_MODELVIEW, GL_PROJECTION и GL_TEXTURE) поэтому чтобы преобразовать матрицу перспективы мы выбираем ее функцией glMatrixMode.

```
;выберем режим вычисления перспективных преобразований (наилучший)
invoke glHint,GL_PERSPECTIVE_CORRECTION_HINT,GL_NICEST
;выберем для преобразований матрицу перспективной проекции
invoke glMatrixMode,GL_PROJECTION
;умножим ее на матрицу перспективы,
;т.е. попросту включим ее (используем макрос glcall т.к.
;параметры передаются в виде 8 байтов)
glcall gluPerspective,90.0,ratio,0.1,100.0
;выберем для преобразований матрицу изображения
invoke glMatrixMode,GL_MODELVIEW
```

Ура, а вот и он, "основной цикл". В нем вы "пока" ;) не увидите навороченной демы от haujobb, просто очистка экрана и выход по Esc. Т.к. мы работаем в режиме "двойной буферизации" то рендеринг идет в дополнительный буфер, для отображения его на экране нужно вызвать функцию GDI32 "SwapBuffers". Дабы впоследствии (когда мы добавим сюда что-нибудь) все не вертелось с бешеной скоростью, добавим сюда еще и синхронизацию с таймером

```
;основной цикл
.draw:
;получаем текущее значение
;счетчика начала работы Windows (для синхронизации)
invoke GetTickCount
;сравним его с сохраненным значением
cmp eax,[msec]
;если оно не изменилось то ждем
jz .draw
;если значение поменялось сохраним его
mov [msec],eax
;очистим буфер экрана
invoke glClear,GL_COLOR_BUFFER_BIT
;отобразим буфер на экран
invoke SwapBuffers,ebp
;проверим на нажатие клавиши ESC
invoke GetAsyncKeyState,VK_ESCAPE
;если она не нажата
test eax,eax
;то продолжим цикл
jz .draw
```

Не забудем выйти и описать переменные ;) Обратите внимание как (как просто ! ;)) в Fasm'e описывается импорт и ресурсы.

```
;выход из программы
invoke ExitProcess,ebx
;заголовок окна
szTitle db 'OpenGL tutorial by Tyler Durden - Simple',0
;имя предопределенного класса окна
szClass db 'edit',0
;включим файл с описанием импорта
data import
include '..\include\imports.inc'
end data
;описание ресурсов
data resource
directory RT_ICON,icons,RT_GROUP_ICON,group_icons
resource icons,1,LANG_NEUTRAL,icon_data
resource group_icons,1,LANG_NEUTRAL,icon
icon icon,icon_data,'..\resources\icons\simple.ico'
end data
;счетчик тиков таймера
msec dd ?
```

```
;соотношение разрешений экрана по горизонтали и вертикали
ratio dq ?
;дескриптор пиксельного формата
pfd PIXELFORMATDESCRIPTOR
```

Ну, вот в принципе и все, в первом OpenGL приложении. Это, типа, был "template", дальше я буду добавлять только измененные части.

Глава 4: Первый цветной треугольник (и тут негры ;))

Ну, покончив с пережитками Win32 программирования, откинувшись (и оттопырившись) на спинку стула, можно наконец приступить к OpenGL'у вплотную... Оказывается нарисовать на нем треугольник не просто, а, как говорится, очень просто ! Все что надо это задать его координаты. Для начала сообщим что мы хотим, собственно, рисовать треугольники (GL_TRIANGLES) т.е. каждые три точки определяют один треугольник (следующие три - уже другой)

```
;начало рисования треугольника
invoke glBegin,GL_TRIANGLES
```

Теперь задаем координаты его вершин и их цвет. Цвет представляет собой 3 float'a (красную, зеленую и синюю (чуть было не написал голубую ;)) составляющие) о чем говорит постфикс функции "3f". Этими префиксами/постфиксами отличаются все функции OpenGL API (т.е. если надо задать еще и alpha-состовляющую - используем функцию glColor4f, если нужно задавать параметры в Double то 3d/4d или 3i/4i для Integer) То же самое относится и к функции glVertex3f (соответственно первый параметр - координата X, второй - Y, третий - Z)

```
;цвет 1-й вершины
invoke glColor3f,1.0,0.0,0.0
;ее координаты
invoke glVertex3f,-1.0,-1.0,1.0
;цвет 2-й вершины
invoke glColor3f,0.0,1.0,0.0
;ее координаты
invoke glVertex3f,1.0,-1.0,1.0
;цвет 1-й вершины
invoke glColor3f,0.0,0.0,1.0
;ее координаты
invoke glVertex3f,1.0,1.0,1.0
```

Все, нарисовались !

```
;конец рисования треугольника
invoke glEnd
```

Если все так и оставить, то мы ничего не увидим, т.к. треугольник будет лежать прямо в точке наблюдения, надо бы его отодвинуть. Но, так как мы будем делать это каждую итерацию цикла то наш треугольник будет все дальше и дальше отодвигаться от нас (проверьте это сами, удалив вызов glLoadIdentity) поэтому вначале мы будем "обнулять" матрицу изображения.

```
;обнулим текущую матрицу (матрицу изображения)
invoke glLoadIdentity
;отодвинем объекты в глубь экрана (z=-3.5)
invoke glTranslatef,ebx,ebx,-3.5
```

Просто нарисованный треугольник ничем не говорит нам о своей "трехмерности", надо его для этого повращать. Делается это функцией glRotate которая вращает (умножает текущую матрицу на матрицу поворота) на угол [theta] вокруг векторов с координатами (0,0,0) и (второй параметр, третий параметр, четвертый параметр) соответственно. Угол мы будем увеличивать каждую итерацию цикла (т.к. матрица изображения "обнуляется")

```
;умножим матрицу изображения на матрицу поворота
;(повернем все объекты сцены
```

```
;на угол theta относительно
;вектора из (0.0,0.0,0.0) в (0.0,0.0,1.0))
invoke glRotatef,[theta],ebx,ebx,1.0
. . .
;загрузим значение угла theta
fld [theta]
;увеличим его на значение delta
fadd [delta]
;и запишем обратно
fstp [theta]
```

Вот и все ! И оч-ч-чень просто, по-моему...

Глава 5: Да будет свет !

Свет в OpenGL включается очень просто (как и многое другое) выключателем ;) т.е. функцией glEnable с параметром GL_LIGHTING (отключается соответственно функцией glDisable). Нужно еще включить еще и определенный источник GL_LIGHT* (номер источника). Мы не будем пока парить себе мозг разнообразными видами источников и их параметрами, включим для начала источник GL_LIGHT0 (для него параметры настроены по умолчанию)

```
;включим источник света GL_LIGHT0 (используя значения по умолчанию)
invoke glEnable,GL_LIGHT0
;включим освещение
invoke glEnable,GL_LIGHTING
```

Так как в этой главе мы будем рисовать объемную фигуру (кубик) то нужно еще включить режим отсечения не лицевых граней (а то все грани кубика будут рисоваться и перекрывать друг друга) т.е. z-буфер. Делается это опять же функцией glEnable. Цвет получившегося (освещенного) полигона зависит от свойств "материала" и только :(Чтобы правильно отображать его цвет нужно еще включить режим его отслеживания (GL_COLOR_MATERIAL).

```
;включим режим отсечения не лицевых граней (z-буфер)
invoke glEnable,GL_DEPTH_TEST
;включим изменение свойств материала в зависимости от его цвета
invoke glEnable,GL_COLOR_MATERIAL
```

Для того чтобы использовать освещение, для каждой грани нужно определить нормаль. Делается это функцией glNormal3f, координата X, координата Y, координата Z

```
;начало рисования куба из 6 четырехугольников GL_QUADS
;(каждые 4 точки glVertex описывают один четырехугольник)
invoke glBegin,GL_QUADS
;нормаль 1-го четырехугольника
invoke glNormal3f,0.0,0.0,1.0
;цвет 1-й вершины
invoke glColor3f,1.0,0.0,0.0
;ее координаты
invoke glVertex3f,-1.0,-1.0,1.0
;цвет 2-й вершины
invoke glColor3f,0.0,1.0,0.0
;ее координаты
invoke glVertex3f,1.0,-1.0,1.0
;и т.д.
invoke glColor3f,0.0,0.0,1.0
invoke glVertex3f,1.0,1.0,1.0
invoke glColor3f,1.0,1.0,0.0
invoke glVertex3f,-1.0,1.0,1.0
;нормаль 2-го четырехугольника
invoke glNormal3f,0.0,0.0,-1.0
invoke glColor3f,0.0,0.0,1.0
invoke glVertex3f,-1.0,-1.0,-1.0
invoke glColor3f,0.0,1.0,0.0
invoke glVertex3f,-1.0,1.0,-1.0
invoke glColor3f,1.0,0.0,0.0
```

```
invoke glVertex3f,1.0,1.0,-1.0
invoke glColor3f,1.0,1.0,0.0
invoke glVertex3f,1.0,-1.0,-1.0
;нормаль 3-го четырехугольника
invoke glNormal3f,0.0,1.0,0.0
invoke glColor3f,0.0,1.0,0.0
invoke glVertex3f,-1.0,1.0,-1.0
invoke glColor3f,1.0,1.0,0.0
invoke glVertex3f,-1.0,1.0,1.0
invoke glColor3f,0.0,0.0,1.0
invoke glVertex3f,1.0,1.0,1.0
invoke glColor3f,1.0,0.0,0.0
invoke glVertex3f,1.0,1.0,-1.0
;нормаль 4-го четырехугольника
invoke glNormal3f,0.0,-1.0,0.0
invoke glColor3f,0.0,0.0,1.0
invoke glVertex3f,-1.0,-1.0,-1.0
invoke glColor3f,1.0,1.0,0.0
invoke glVertex3f,1.0,-1.0,-1.0
invoke glColor3f,0.0,1.0,0.0
invoke glVertex3f,1.0,-1.0,1.0
invoke glColor3f,1.0,0.0,0.0
invoke glVertex3f,-1.0,-1.0,1.0
;нормаль 5-го четырехугольника
invoke glNormal3f,1.0,0.0,0.0
invoke glColor3f,1.0,1.0,0.0
invoke glVertex3f,1.0,-1.0,-1.0
invoke glColor3f,1.0,0.0,0.0
invoke glVertex3f,1.0,1.0,-1.0
invoke glColor3f,0.0,0.0,1.0
invoke glVertex3f,1.0,1.0,1.0
invoke glColor3f,0.0,1.0,0.0
invoke glVertex3f,1.0,-1.0,1.0
;нормаль 6-го четырехугольника
invoke glNormal3f,-1.0,0.0,0.0
invoke glColor3f,0.0,0.0,1.0
invoke glVertex3f,-1.0,-1.0,-1.0
invoke glColor3f,1.0,0.0,0.0
invoke glVertex3f,-1.0,-1.0,1.0
invoke glColor3f,1.0,1.0,0.0
invoke glVertex3f,-1.0,1.0,1.0
invoke glColor3f,0.0,1.0,0.0
invoke glVertex3f,-1.0,1.0,-1.0
;конец рисования куба
invoke glEnd
```

Заметьте что мы вызываем glRotatef 3 раза, для придания траектории вращения большей random'ности

Глава 5: Текстурирование

Для пущего реализЬма напялим на наш кубик текстуру. Делается это тоже проще пареной репы. Для начала нужно ее (текстуру) нарисовать (можно выдрать откуда-то, главное помните, что ее размер должен быть степенью двойки (256*256*24bpp в примере) ;), и сохранить в формате "raw" (например Photoshop'ом или IrfanView) конечно размер полученного файла будет немаленьким, но проще потом exe-шник сжать UPX'ом чем париться с JPEG/TGA форматами... Затем подключим полученный файл в виде ресурса в resourse.inc

```
fileres texture,'textures.raw'
```

Далее ее нужно перевести в понятный OpenGL'у формат функцией glTexImage2D. GL_TEXTURE_2D говорит что текстура у нас 2-х мерная, следующий ноль (ebx) определяет mip-map уровень для нее (для ускорения текстурирования из текстуры можно предварительно сгенерировать несколько уменьшенных копий, но у нас пока только одна), 3-ка - это число

компонент в цвете (у нас RGB, а может быть и1-а (R), 2-е (RA), 4-е (RGBA)), затем размерность текстуры по x и y соответственно (256*256), "0" (ebx) это толщина "рамки" вокруг текстуры, далее формат текстуры (RGB, RGB, RGB, ...), тип каждой компоненты (у нас байт) и, собственно, смещение в файле (texture+16).

```
;создадим 256x256x24bpp текстуру
;из ресурса texture (raw файл)
invoke glTexImage2D,GL_TEXTURE_2D,ebx,\
  3,256,256,ebx,GL_RGB,\
  GL_UNSIGNED_BYTE,texture+16
```

Теперь осталось определить еще некоторые параметры текстуры, такие как фильтры уменьшения/увеличения (GL_TEXTURE_MIN_FILTER/GL_TEXTURE_MAG_FILTER) (т.е. которые применяются когда текстура, соответственно, меньше/больше требуемой для текстурирования в данный момент времени) функцией glTexParameteri. Установим их в GL_LINEAR (лучшее качество, можно еще GL_NEAREST побыстрее, но похуже)

```
;установим фильтр текстуры при уменьшении (linear)
invoke glTexParameteri,GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR
;установим фильтр текстуры при увеличении (linear)
invoke glTexParameteri,GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR
```

Вообще говоря, нужно было предварительно выбрать ID текстуры (1 для первой текстуры, 2 для второй и т. д.) функцией glBindTexture, но т.к. у нас только одна, и вся работа идет с 1-ой текстурой (по умолчанию), то нам ничего не мешает этого не делать.

```
;выберем текстуру номер 1
invoke glBindTexture,GL_TEXTURE_2D,1
```

Осталось только включить текстурирование функцией glEnable

```
;включим текстурирование
invoke glEnable,GL_TEXTURE_2D
```

Усе, однако OpenGL не знает как привязать координаты вершин объектов к текстурным координатам (за исключением автоматической генерации текстурных координат, но об этом позже) так что надо ей об этом сообщить функцией glTexCoord2f с двумя параметрами (для 2-х мерной текстуры) x и y координатами текстуры (они изменяются также как и все остальное в OpenGL - в относительных координатах от 0.0 до 1.0)

```
;нормаль 1-го четырехугольника
invoke glNormal3f,0.0,0.0,1.0
;координата текстуры для 1-ой вершины
invoke glTexCoord2f,0.0,1.0
;1-я вершина
invoke glVertex3f,-1.0,-1.0,1.0
;координата текстуры для 2-ой вершины
invoke glTexCoord2f,1.0,1.0
;2-я вершина
invoke glVertex3f,1.0,-1.0,1.0
;и т.д.
```

В случае с несколькими текстурами, их выбор осуществляется, как я уже говорил, функцией glBindTexture.

Глава 6: Alpha смешивание (blending) или шара продолжается

Alpha смешивание позволяет нам использовать эффекты прозрачности. Для этого нужно опять таки его просто включить функцией glEnable (вам даже не надо знать как оно работает ! Вот шара :) помучившись со своим дос-софтверным 3D движком можно вздохнуть спокойно ;)), настроить его тип можно функцией glBlendFunc, параметры источника (что смешивается), параметры результата (с чем смешивается). Параметры зависят от требуемого результата.

```
;выберем режим работы альфа смешивания
invoke glBlendFunc,GL_SRC_ALPHA,GL_ONE
;и включим его
invoke glEnable,GL_BLEND
```

Чтобы было с чем смешивать, нужно отключить режим отсечения не лицевых граней (попросту z-буфер) функцией glDisable.

```
;выключим режим отсечения не лицевых граней (z-буфер)
invoke glDisable,GL_DEPTH_TEST
```

либо вообще его не включать (как у нас в примере).

Глава 7: Туман (без лошади)

А теперь "офигенно" сложный этап tutorial'a ;) Туман ! Делается даже сложней чем alpha смешивание ;) а именно:

```
;выберем режим работы тумана (экспоненциальный)
invoke glFogi,GL_FOG_MODE,GL_EXP2
;установим его цвет (fogColor)
invoke glFogfv,GL_FOG_COLOR,fogColor
;его интенсивность
invoke glFogf,GL_FOG_DENSITY,0.55
;начало
invoke glFogf,GL_FOG_START,1.0
;и конец
invoke glFogf,GL_FOG_END,3.0
;установим режим вычисления тумана (наилучший)
invoke glHint,GL_FOG_HINT,GL_NICEST
;установим цвет очистки экрана равный цвету тумана
invoke glClearColor,0.5,0.5,0.5,1.0
;включим туман
invoke glEnable,GL_FOG
. . .
;цвет тумана
fogColor dd 0.5,0.5,0.5,1.0
```

И все ! Параметры я думаю говорят сами за себя... Так как "туманятся" только объекты, то для пущей выразительности окрашиваем остатки экрана под цвет тумана (непонятно почему разработчики OpenGL не предусмотрели для glClearColor вариант с постфиксом fv (передается указатель на массив из Float'ов). Режим "работы" тумана можно выбрать и другой (ну, например, с 9 до 6 вечера, без выходных ;)) GL_EXP или GL_LINEAR, поэкспериментируйте...

Глава 8: Outline шрифты

Заценим теперь, какую "титаническую" работу проделала Microsoft (ну вот, а говорил, что не будет про них) в своем scream-saver'e "3D Text". Для начала привычные для шрифтов вызовы Win32 API: создаем объект типа (ти-и-ипа) "шрифт" (все значения оставляем на совести Windows), и даем установку (а-ля Кашпировский) для "рисования".

```
;создадим шрифт
```

```
;(все значения по умолчанию "0", шрифт семейства "Comic Sans MS")
invoke CreateFont,ebx,ebx,ebx,ebx,ebx,ebx,\
 ebx,ebx,ebx,ebx,ebx,ebx,ebx,fontName
;выберем полученый объект (шрифт) для текущего контекста окна (ebp)
invoke SelectObject,ebp,eax
```

Дальше вступает в дело OpenGL: преобразовывает векторный (именно векторный ! Шрифт "System" преобразовать не удастся, а жаль) в так называемый список (с... что ?!), т.е. последовательность OpenGL команд описания графических примитивов (те что обычно задаются между glBegin, glEnd). Для этого используется функция wglUseFontOutlinesA (или ее unicode версия wglUseFontOutlinesW для символов с кодами больше 255) Получаем "outline font". Мы будем преобразовывать символы начиная с нулевого (ebx) по 255 (всего, значит, 256 символов), единица означает что используются полигоны, а не линии ("0"), "0.2" это толщина нашего шрифта, в ebp, если помните, сохранен контекст OpenGL. Для операций с несколькими (у нас - 256) списками нужно выбрать первый список (вообще-то его их нужно было сгенерировать функцией glGenLists, но за нас это делает OpenGL). Для этого используется функция glListBase.

```
;преобразуем символы от 0-го до 256-го
;в список 3-х мерных моделей (outline font)
;толщиной 0.2.
;"1" означает использование полигонов вместо линий ("0")
invoke wglUseFontOutlinesA,ebp,\
 ebx,256,listBase,ebx,0.2,1,ebx
;выберем первый элемент в списке
invoke glListBase,listBase
```

Посмотрим на предыдущие примеры tutorial'a, и офигеем ;) Зачем было столько носиться с оптимизацией, если размер какой-то завшивленной 256*256*24bpp текстуры занимает целых 200 кило ? Это не есть хорошо, будем теперь генерить ее вручную ! Это не tutorial про генерацию текстур (читайте в "Hugi" про это), поэтому выберем простую "шумовую" текстуру, для этого неплохо подходят код нашей проги. Заюзаем его в качестве текстуры (хе-хе, вот это я понимаю, извращение). OpenGL, конечно, ничего не знает про координаты текстуры для "outline" шрифта, да и вклиниться в работу функции wglUseFontOutlines и "наследить" там вызовами glTexCoord2f будет сложно (для этого нужно юзать feedback буфер), так что просто предоставим заниматься этим самой OpenGL, включив автоматическую генерацию текстурных координат (да, да, вот где еще одна шара !). Причем выберем режим текстуры GL_SPHERE_MAP, который обычно используется в создании всяческих отражений (enironment mapping).

```
;создадим 16x16x32bpp текстуру используя
;в качестве значений цветов пикселей опкоды нашей программы
;(начиная с метки start)
invoke glTexImage2D,GL_TEXTURE_2D,ebx,4,16,\
  16,ebx,GL_RGBA,GL_UNSIGNED_BYTE,start
;установим тип генерируемой текстуры
;в направлении "S" (GL_SPHERE_MAP)
invoke glTexGeni,GL_S,GL_TEXTURE_GEN_MODE,GL_SPHERE_MAP
;в направлении "T" (GL_SPHERE_MAP)
invoke glTexGeni,GL_T,GL_TEXTURE_GEN_MODE,GL_SPHERE_MAP
;и включим автоматическую генерацию
;текстурных координат в направлении "S"
invoke glEnable,GL_TEXTURE_GEN_S
;и "T"
invoke glEnable,GL_TEXTURE_GEN_T
```

Ндааа... Опять глава кончилась не успев начаться ;) Ах да, нужно еще написать какое-нибудь слово (желательно не из трех букв, хотя в OpenGL слава Богу еще не встроили Parental Lock ;) все-таки это не DirectX), для этого вызовем списки (хмм, с этим словом главное не описАться... ой) с номерами равными кодам желаемых букв (т.к. мы генерили символы от "0" до "255"). Делаем это функцией glCallLists.

```
;вызавем txtIntro_length элементов списка (шрифта)
;с номерами указанными в строке байт (GL_UNSIGNED_BYTE) txtIntro
```

```
invoke glCallLists,txtIntro_length,GL_UNSIGNED_BYTE,txtIntro
```

Ура ! Получилось довольно красиво (и, главное, всего на три кило). Можно даже забацать из этого какое-нибудь "Newschool scrolltro". Дерзайте.

Глава 9: Motion blur и рендеринг в текстуру

И так, мы уже вплотную подобрались к алгоритмам используемым в демках. Т.к. OpenGL не позволяет (нормальным образом) получить доступ к видеопамяти, поэтому и алгоритмы эффектов здесь соответствующие - извращенческие. Так что обычным образом motion blur нам сделать не удастся, будем юзать "встроенную" возможность OpenGL blur'ить текстуры при их растягивании (ну да, нам даже не надо знать как работают motion blur фильтры !) Вкратце план довольно прост:

1. Отрендерив сцену в окно размером blurSize*blurSize (512*512 в примере), скопируем полученное изображение из экранной памяти в текстуру.
2. Идем дальше ("Строго на север, в порядке 50 метров"), нарисуем четырехугольник с полученной текстурой, размером с экран, поверх изображения.
3. Затем ("Проходя мимо пЫхты"), отрисуем сцену еще раз.

Все, т.к. альфа смешивание включено, то каждый раз смешивая текстуру с предыдущими сценами (с альфа компонентой меньше "1.0") и черный экран (с альфа компонентой "0.0"), она будет постепенно "исчезать". Так как мы будем рендерить все в пустую текстуру, то с начала нужно ее создать, делается это как и обычно:

```
;выберем текстуру номер 1
invoke glBindTexture,GL_TEXTURE_2D,1
;создадим blurSizexblurSizex24bpp
;текстуру из буфера TextureBlank (пустого)
invoke glTexImage2D,GL_TEXTURE_2D,ebx,3,blurSize,\
 blurSize,ebx,GL_RGB,GL_UNSIGNED_BYTE,textureBlank
```

Заметьте, что кубик мы предварительно "компилируем" в список.

```
;создадим новый список cubeList
invoke glNewList,cubeList,GL_COMPILE
```

Blur'ить мы будем не каждый раз, а только через 4 фрейма (иначе "хвост" будет слишком маленьким, да и тормознуто это).

```
;увеличим счетчик кадров blur'a
inc [blurCounter]
;он стал равен 4-м ?
cmp [blurCounter],4
;нет не в этот раз ;)
jnz .notBlur
```

С начала нужно установить окно размером blurSize*blurSize, и отрендерить туда нашу "сцену".

```
;установим окно вывода с
;координатами 0,0 размером blurSize*blurSize
invoke glViewport,ebx,ebx,blurSize,blurSize
;отрендерим blur сцену
call renderMotionBlur
```

Скопируем то что получилось в текстуру, функцией glCopyTexImage2D. GL_TEXTURE_2D показывает что текстура двухмерная, следующий ноль (ebx) - это mip-map уровень, затем координаты текстуры (0,0), ее размер (blurSize*blurSize), и размер границы ("0"). Очистим экран, и вернем OpenGL'ое окно обратно.

```
;выберем текстуру номер 1
invoke glBindTexture,GL_TEXTURE_2D,1
;скопируем в нее отрендеринную только что
;сцену размером blurSize*blurSize
```

```
invoke glCopyTexImage2D,\
 GL_TEXTURE_2D,ebx,GL_RGB,ebx,ebx,blurSize,blurSize,ebx
;очистим буфер экрана и z-буфер
invoke glClear,GL_COLOR_BUFFER_BIT+GL_DEPTH_BUFFER_BIT
;установим окно вывода с координатами 0,0
;размером screenWidth*screenHeight
invoke glViewport,ebx,ebx,[screenWidth],[screenHeight]
```

Теперь как нам нарисовать четырехугольник точно размером с экран ? Ведь OpenGL'у плевать на пиксели, он оперирует какими-то "условными единицами" (зелеными, наверное ;)). Для этого нам надо установить так называемую "Ортогональную (прямоугольную) проекцию" делается это не сложнее установки перспективной проекции:

```
;и умножим на матрицу ортогональной проекции
glcall glOrtho,0.0,screenWidthDouble,screenHeightDouble,0.0,-1.0,1.0
```

Усе, рисуем нашу сцену четырехугольником (заметьте, что перспективную проекцию мы устанавливаем не функцией, а, просто, сохранив проекционную матрицу в стеке функцией glPushMatrix, и восстанавливаем ее функцией glPopMatrix)

```
;выберем текстуру номер 1
invoke glBindTexture,GL_TEXTURE_2D,1
;уменьшим альфа компоненту объектов (0.9)
invoke glColor4f,1.0,1.0,1.0,0.9
;выберем для преобразований матрицу перспективной проекции
invoke glMatrixMode,GL_PROJECTION
;сохранить текущую матрицу (матрицу перспективы)
invoke glPushMatrix
;обнулим ее
invoke glLoadIdentity
;и умножим на матрицу ортогональной проекции
glcall glOrtho,0.0,screenWidthDouble,screenHeightDouble,0.0,-1.0,1.0
;выберем для преобразований матрицу изображения
invoke glMatrixMode,GL_MODELVIEW
;обнулим текущую матрицу (матрицу изображения)
invoke glLoadIdentity
;нарисуем четырехугольник размером в экран
;(отрендеренная до этого сцена используется в качестве тексуры)
invoke glBegin,GL_QUADS
invoke glTexCoord2f,ebx,1.0
invoke glVertex2i,ebx,ebx
invoke glTexCoord2f,ebx,ebx
invoke glVertex2i,ebx,[screenHeight]
invoke glTexCoord2f,1.0,ebx
invoke glVertex2i,[screenWidth],[screenHeight]
invoke glTexCoord2f,1.0,1.0
invoke glVertex2i,[screenWidth],ebx
invoke glEnd
;выберем для преобразований матрицу перспективной проекции
invoke glMatrixMode,GL_PROJECTION
;восстановим ее из стека
invoke glPopMatrix
```

В принципе ничего сложного тут нет, все абсолютно "прозрачно", смотрите код и все поймете.

Глава 10: Zoom blur

Ур-ра ! Последняя часть tutorial'a (пока, гы гы). Надоели вы мне, хех, как сказал один доцент (тоже ученый, у него 3 класса образования): "пришить бы вас, да возиться неохота" ;). Но пока, так... Копаем... Копать будем "zoom blur", или как его еще иногда (и неправильно) называют "radial blur", или даже "volumetric lights"... Но не в названии дело, а в его простоте. Собственно, после предыидущей главы, делать мне тут нечего, разве что так, "на шухере постою". Алгоритм довольно тормознутый (для моей отстойной видюхи GeForce4 MX), в 1024*768 лучше не пускать, поэтому установим-ка, для начала, что нибудь вроде 640*480:

```
;подготовим структуру описывающую видеорежим, ее размер
mov [dmScreenSettings.dmSize],dmScreenSettings_size
```

```
;разрешение по горизонтали
mov [dmScreenSettings.dmPelsWidth],screenWidth
;вертикали
mov [dmScreenSettings.dmPelsHeight],screenHeight
;изменяемые поля
mov [dmScreenSettings.dmFields],DM_PELSWIDTH+DM_PELSHEIGHT
;сменим видеорежим
invoke ChangeDisplaySettings,dmScreenSettings,ebx
```

Обратите внимание на, уже набивший всем оскомину, "эффект" широкоформатной пленки (типа, фи-и-и-ильму, фи-и-и-ильму !), который кое-кто уже умудрился впарить даже в 256b intro. Т.к. я поубивал много лишних вызовов процедур (к примеру, glViewport нужно было бы вставить после инициализации), то прога потеряла некоторую долю логичности (а заодно и пару "килограмм"), но к 10 главе мы уже все стали "хавцами" в OpenGl'e, неправда ли ?

```
;установим окно вывода с координатами 0,77
;размером screenWidth*screenHeight
invoke glViewport,ebx,77,screenWidth,screenHeight
```

Сам алгоритм, который очень похож на motion blur, состоит в следующем (неизмененные места оставлены специально):

1. Отрендерив сцену в окно размером blurSize*blurSize (256*256 в примере), скопируем полученное изображение из экранной памяти в текстуру.
2. Нарисуем четырехугольники с полученной текстурой поверх изображения несколько (blurCount) раз, каждый раз увеличивая их размер, от blurSize до полноэкранного размера (640), и уменьшая значение альфа компоненты.
3. Рулим ;) (можно отрендерить сцену еще раз, поверх этого, но мы не будем).

Смотрим код:

```
;инициализируем максимальный радиус радиального blur'a
mov ecx,blurCount
;и начальные значения переменных
mov dword [blurInc],ebx
mov dword [blurDec],1.0
mov dword [blurAlpha],0.35
;цикл радиального размытия
.renderMotionBlur:
;сохраним счетчик
push ecx
;увеличим радиус "справа"/уменьшим "слева"
;и уменьшим альфа компоненту радиального blur'a
fld dword [blurDelta]
fld st0
fadd dword [blurInc]
fstp dword [blurInc]
fsubr dword [blurDec]
fstp dword [blurDec]
fld dword [blurAlpha]
fsub dword [alphaDelta]
fstp dword [blurAlpha]
;выберем текущее значение альфа компоненты
invoke glColor4f,[esp],[esp],1.0,[blurAlpha]
;нарисуем четырехугольник размером в экран
;(отрендеренная до этого сцена используется в качестве тексуры)
invoke glBegin,GL_QUADS
invoke glTexCoord2f,[blurInc],[blurDec]
invoke glVertex2i,ebx,ebx
invoke glTexCoord2f,[esp],[blurInc]
invoke glVertex2i,ebx,screenHeight
invoke glTexCoord2f,[blurDec],[blurInc]
invoke glVertex2i,screenWidth,screenHeight
invoke glTexCoord2f,[esp],[blurDec]
invoke glVertex2i,screenWidth,ebx
invoke glEnd
;восстановим счетчик
pop ecx
```

```
;уменьшим его
dec ecx
;и продолжим цикл
jnz .renderMotionBlur
```

Незабудем вернуть видеорежим обратно, для этого просто вызовем функцию "ChangeDisplaySettings(null,0)"

```
;вернем видеорежим обратно
invoke ChangeDisplaySettings,ebx,ebx
```

Вот и все ! Уже украли/обрили... Просто, но, черт возьми, эффЭктно (и впереди я, на белом верблюде). Так о чем это я ? Ага, пришло, наконец, время долгожданного прощания. Просьба не устраивать сцен и истерик ;) обнимемся как братья... Э-э-э-э, что там еще говорят ? Желаю счастья в личной жизни... Один маленький, но очень гордый птичка... Нет, не то... Типа, надобранич дитлохи, до зустричи (тут я прям прослезился, как все Хрюши, Степаши и тети Тани/Амаяки Акопяны, канала О.Р.Т., вместе взятые)...

Приложение 1: Vertex arrays

Не успев обрадоваться от вышеописанного расставания (сцена с рыдающими детьми: "Папа ! Папа ! На кого ты нас покинул ?!"). Люба, я вернулся (wmplayer.exe c:\windows\media\tada.wav). Ну "да-ра-гггие мАи" ну не мог же я выпустить вас "в люди" голыми, так сказать ! Так бы сейчас и писали glBegin'ами. Шо вы ? Это уже совсем не модно ! Щас "в Европе, а также в лучших домах Филадельфии" все поголовно юзают массивы вершин. Вообще то, массив вершин (vertex array) - это почти тоже самое, что и команды между glBegin/glEnd, только их выполняет сама карта, подставляя значения из нашего массива. Как нам уже известно, описывать приходится не только, собственно, вершины, но и нормали, текстурные координаты, цвет и пр., поэтому и массивы бывают разные. Однако, мы не будем мучить себя подключением всех массивов, скинем просто все в кучу функцией "glInterleavedArrays" (т.е. данные в массиве идут последовательно, сначала нормаль одной вершины, потом ее текстурные координаты, цвет и пр. затем сами координаты вершины. Затем тоже самое для следующей вершины и т.д.). Она принимает параметрами тип "сваленных в кучу" массивов, тут это GL_N3F_V3F, т.е. N3F - массив нормалей, по 3 float'а на нормаль и V3F - массив вершин, тоже из 3-х float'ов на вершину, так называемый "stride" - "расстояние" между данными для соседней вершины (у нас будет "0" (ebx) т.к. данные в массиве идут подряд), и ,наконец, ссылку на массив с данными (mdlSheep). Но glInterleavedArrays только подключает массивы на отображение, их еще нужно, собственно, отобразить. Делается это функциейй "glDrawArrays". Первый параметр - это тип используемых примитивов (GL_TRIANGLES), второй - это первый элемент массива (ebx=0, т.е. рисуем все элементы) и третий параметр - это общее число элементов...

```
;включим отображение массивов нормалей
;(по 3 float'а на нормаль) и вершин (3 float'а на нормаль)
invoke glInterleavedArrays,GL_N3F_V3F,ebx,mdlShip
;отрисуем вершинный массив из 8686 треугольников,
;начиная с треугольника номер "0" (ebx)
invoke glDrawArrays,GL_TRIANGLES,ebx,8686*3-1
```

Дело за малым - осталось найти де взять этот массив ;) Можно, конечно, и ручками (как в следующем приложении), но я (для демонстрации "крутизны" массивов вершин) отконвертил модельку X-Wing'а (предварительно сохранив ее 3DStudio в формате "ASC"). Можно было вставить ее просто как переменную (mdlShip dd ...) но это б заняло полтора мегабайта ;) Так что пришлось вынести ее в отдельный файл, и скомпилить FASM'ом, в результате чего получился 600 Kb бинарник, который я засунул в ресурсы директивой "file".

```
mdlShip file '..\resources\models\ship.bin'
```

Обратите внимание как я "генерил" текстуру (так вот откуда Спилберг спер терминатора T1000 ;)) взяв за основу только красные компоненты "пикселей" (а на деле опкоды... красные уже и сюда добрались...) "GL_LUMINANCE_ALPHA". Для придания текстуре яркости, я использовал специальный режим обработки bitmap'ов (о них в следующем приложении) т.е. здесь - текстуры: "glPixelTransferf" с параметром "GL_RED_SCALE" (множителем красной (нет, ну хоть в кавычках это слово пиши) компоненты) и значением "10.0".

```
invoke glPixelTransferf,GL_RED_SCALE,10.0
```

Ну, теперь все быстренько на борьбу с кубиками и торами ! Рисуйте свои модельки и вперед. Помните только (неоднократно натыкался на такое "недопонимание") что массив индексов, это на самом деле массив индексов цветов в режимах с маленькой глубиной цвета (256 цветов и т.д.) а не массив индексов вершин (как в 3D моделях).

Приложение 2: Битовые карты (ну там, тройка, семерка, туз...)

Ну почему обязательно карты ? Есть много интересных и полезных спортивных игр ! Например литрбо... оуэ-э-э-э битмап шрифты ! Еще бы, как же иначе мы напишем в нашу демку кучу псевдо-философского бреда, и причем непременно (непреме-е-енно !) "разъежающейся-блендещайся" Tahoma'ой... После главы про outlined шрифты, здесь уже все прозаично. Как обычно "создадим" шрифт:

```
;создадим шрифт (размером "9", остальные значения по умолчанию "0"
;, шрифт семейства "Lucida Console")
invoke CreateFont,-9,ebx,ebx,ebx,ebx,\
;ebx,ebx,ebx,ebx,ebx,ebx,ebx,ebx,fontName
;выберем полученый объект
;(шрифт) для текущего контекста окна (ebp)
invoke SelectObject,ebp,eax
```

Теперь, по анологии с outlined шрифтами, преобразуем все это в список команд рисования битмап'ов функцией "wglUseFontBitmaps(контекст OpenGL (ebp),первый символ (ebx),число символов (256),список (берем 0-й))" :

```
;преобразуем символы от 0-го до 256-го в список битмап'ов
invoke wglUseFontBitmapsA,ebp,ebx,256,ebx
```

Установим ортогональную проекцию (для битмап'ов работать в ней удобнее). Вот тут начинается самое интересное... Будем делать "Матрицу" ! Ну, ту самую, что "has Neo...". Надеюсь ему понравилось, но мы за него ОТОМСТИМ ! Вощ-щ-щем расклад такой:

1. Заведем массив на каждую "строку" "матрицы" с координатами X (dword), Y (dword) и ее начальным символом
2. Генерим каждой все вышеописанное
3. В основном цикле распихиваем "строки" по сгенереным координатам, каждый раз начиная с начального символа "строки" и "начального цвета" (черного), увеличивая их для каждой следующей буквы
4. Начальный символ увеличивается для каждой "charChangeFrequency" "строки"
5. Каждый последний символ "строки" выделяем голубым цветом (мстим за Нео)
6. Так как координата "Y" увеличивается на размер "строки" по вертикали, то вернем ее на место, пришьем, попутно, + 3 за побег (+ 5 за детсад), т.к. "строки" должны "бежать" вниз
7. Если какая-нибудь "строка" сЦучилась (читай, вылезла за края экрана) - обнулим ей "Y" (шоб неповадно было)
8. Просто матрица это ж страшный "порожняк", поэтому напялим эту в качестве текстуры, для начала, хотяб на кубик (месть продолжается). Делается это как обычно, копированием экрана в текстуру функцией "glCopyTexImage2D"

Надеюсь вы что-нибудь поняли... Если нет, то смотрите код, там понятней. Вообще-то это была задумка написать "город" а-ля в "Матрице", отсюда и кубик, но меня потом в конец обломало :(Пишите сами, в общем. На том и сказке/тьюториалу конец (делу венец и т.п.) а кто слушал #$%&@# !!!

А у вас сколько мониторов? (DirectX)

Автор: SolidCode

Дата: 2005-08-04

исходники к статье (находится в архиве `wasm_files.zip`: `files/0383/MultimonDX.zip`)

Да, это снова SolidCode со своими мониторами. Очень уж меня интересует эта тема. В предыдущей статье мы разбирались, какую информацию о мониторах можно получить средствами Win32 API. Тогда я наивно полагал, что DirectX – это рулез, потому что глубже работает с видеодрайверами и железом.

Но меня мелкомягкие сильно разочаровали. Оказалось, что с помощью DirectX можно узнать ещё меньше информации по сабджу. Хотя и появляется кое-что дополнительное. Но расскажу обо всём по порядку. Эта статья предполагает, что вы прочитали предыдущую статью и знаете общую логику работы с мониторами в системе Windows . Исследования для этой статьи проводились в рамках DirectX 8.1. Большое спасибо keyMax за прекрасные статьи и переведённые инклюды. Более мелким шрифтом будет даваться информация по некоторым вопросам, которые изменились в DirectX от версии 8.1 до версии 9.

Для инициализации DirectX 8.1 требуется выполнить следующее:

```
invoke  Direct3DCreate8,D3D_SDK_VERSION
mov     pd3d,eax
```

Если функция вернула нормальный указатель на интерфейс, то начинаем исследование ситуации с видеоадаптерами/мониторами. Надо сказать, что DirectX более интересуется видеоадаптерами. А монитор уж там чего-нибудь покажет. В отличие от GDI , здесь можно узнать количество видеоадаптеров заранее. Вот объявление метода из MSDN.

```
UINT GetAdapterCount();
```

И его вызов средствами MASM32.

```
d3d8    GetAdapterCount,pd3d
mov     nAdapters,eax
```

Видимо DirectX каждому реальному монитору сопоставляет свою логическую «копию» видеоадаптера, что и определяет возвращаемое значение. При ОДНОМ физическом видеоадаптере и двух реальных мониторах мы получаем значение 2. Да, два видеоадаптера в системе, ни больше и не меньше.

Теперь в цикле будем перебирать адаптеры по номерам и получать доступную информацию в структуру D3DADAPTER_IDENTIFIER8 . Перебор начинаем с 0, что есть D3DADAPTER_DEFAULT . Это первичный адаптер/монитор. Он существует в любом случае. Последним будет адаптер под номером nAdapters -1 . Обычно это значение не превышает двух. Информацию о каждом мониторе получаем с помощью метода GetAdapterIdentifier . Вот его объявление из MSDN.

```
HRESULT GetAdapterIdentifier(
  UINT Adapter,
  DWORD Flags,
  D3DADAPTER_IDENTIFIER8* pIdentifier
);
```

Далее следует пример кода в MASM 32, который циклом WHILE обрабатывает все адаптеры. Номер адаптера содержится в регистре ESI . Метод должен возвращать значение D3D_OK , если всё нормально и информация получена.

```
xor     esi,esi
mov     nAdapters,eax
```

```
.while esi<nAdapters
    lea     edx,[ebx].AdapterInfo
    mov     [ebx].nAdapterOrdinal,esi
    d3d8    GetAdapterIdentifier, pd3d, esi, D3DENUM_NO_WHQL_LEVEL, edx
    .break.if eax!=D3D_OK
    ...
    inc     esi
    add     ebx,sizeof DXMON_INFO
.endw
```

Этот метод предоставляет нам максимум интересующей нас информации, которую можно выжать из DirectX. При этом я рекомендую использовать флаг D3DENUM_NO_WHQL_LEVEL EQU 2. Это запрещает функции зависать надолго разыскивая бесполезные подписи мелкомягких (WHQL расшифровывается Windows Hardware Quality Labs). У меня функция вообще вылетала в WinXP при использовании без этого флага. A OllyDbg ругался на кучу дополнительных потоков, которые она порождала и забывала убивать. Но использование этого флага рулез, имхо.

Самое интересное – это содержимое структуры D3DADAPTER_IDENTIFIER8.

```
MAX_DEVICE_IDENTIFIER_STRING equ 512

D3DADAPTER_IDENTIFIER8 STRUC
Driver BYTE MAX_DEVICE_IDENTIFIER_STRING DUP (?)
Description BYTE MAX_DEVICE_IDENTIFIER_STRING DUP (?)
DriverVersion DWORD ? ;LARGE_INTEGER
DriverVersionHi DWORD ?
VendorId DWORD ?
DeviceId DWORD ?
SubSysId DWORD ?
Revision DWORD ?
DeviceIdentifier DWORD 4 DUP (?) ;GUID ?
WHQLLevel DWORD ?
D3DADAPTER_IDENTIFIER8 ENDS
```

Давайте посмотрим, что здесь есть интересного. Прежде всего разберём строку Driver. Здесь мы получаем имя основного файла драйвера. Строка Description даёт нормальное имя видеоадаптера, презентабельное для пользователя. Далее следует 64-битное число, представляющее версию драйвера. Оно делится на 4 16-битных числа. В примере, прилагаемом к статье, вы найдёте готовую функцию, превращающую это число в нормальную строку, идентичную той, которую показывает диагностическая утилита DirectX (dxdiag. exe). Следующие 4 поля особой пользы для нас не предоставляют. В SDK о них сказано, что их можно использовать для определения конкретного чипсета, однако там же советуют использовать эти значения с осторожностью. Далее следует DeviceIdentifier в формате GUID. Функция для его превращения в нормальную строку также находится в примере к статье. WHQLLevel всегда будет ноль, если мы используем флаг D3DENUM_NO_WHQL_LEVEL.

Как видите, реально полезной информации мы получили весьма немного. Правда, узнали имя главного файла драйвера и версию драйвера.

Версия 9.

Применимо к методу GetAdapterIdentifier произошло следующее изменение. Убрали флаг D3DENUM_NO_WHQL_LEVEL и поставили флаг D3DENUM_WHQL_LEVEL . Видимо их достали с руганью по поводу лабораторий качества. Теперь, если в соответствующем аргументе передаётся NULL , то WHQL не получают, а если уж сильно захотели, то ставьте флаг D3DENUM_WHQL_LEVEL .

Структура D 3 DADAPTER _ IDENTIFIER тоже немного изменилась. Примечательно, что добавили строку DeviceName, в которой возвращается внутреннее имя дисплея (типа “\.\ DISPLAY 1”) с намёком на то, что дальше сами получайте информацию об устройстве с помощью стандартных API из user 32. dll.

В восьмой версии DirectX мы ещё можем получить текущий видеорежим соответствующего адаптера с помощью метода GetAdapterDisplayMode;

```
HRESULT GetAdapterDisplayMode(
  UINT Adapter,
  D3DDISPLAYMODE* pMode
);
```

Для получения всех доступных видеорежимов используем метод GetAdapterModeCount, чтобы получить количество режимов. А потом в цикле перебираем все эти режимы, начиная с 0. В этом нам поможет метод EnumAdapterModes.

```
UINT GetAdapterModeCount(
  UINT Adapter
);

HRESULT EnumAdapterModes(
   UINT Adapter,
   UINT Mode,
   D3DDISPLAYMODE* pMode
   );
```

В DirectX видеорежимы описываются несколько иначе, чем в GDI. Вот объявление структуры D3DDISPLAYMODE , которая содержит информацию о видеорежиме.

```
D3DDISPLAYMODE STRUC
    Width1  UINT ?
    Height  UINT ?
    RefreshRate UINT  ?
    Format  DWORD ? ;D3DFORMAT
D3DDISPLAYMODE ENDS
```

Глубина цвета здесь описывается не количеством битов для описания цвета, а с помощью специальных форматов поверхностей (D3DFORMAT). Они более информативны, чем количество битов.

Если же вы всё-таки слишком любопытные и хотите узнать всё про мониторы, то используете метод GetAdapterMonitor.

```
HMONITOR GetAdapterMonitor(
  UINT Adapter
);
```

Так мы получим стандартный хэндл монитора, а уже от него всю остальную информацию обычными функциями API, которые были рассмотрены в прошлой статье.

На этом наше повествование заканчивается. Более мне не удалось найти способы получить информацию о видеосистеме компьютера. Правда, в интерфейсах DirectDraw (версии DirectX 7 и меньше) можно ещё получить размер видеопамяти. В версии 8 мелкомягкие, по видимому, решили, что эта информация лишняя. Так что Win32API всё-равно незаменимы.

Успешного творчества.

Извлечение текстурированных 3D моделей из приложений Direct3D

Автор: FrostFix

Дата: 2006-03-16

Исходный код (находится в архиве `wasm_files.zip`: `files/0399/d3d8.zip`)

1. Введение

Целью статьи является исследование принципов, позволяющих извлекать трехмерные текстурированные модели из приложений Direct3D.

Задача: Разработать и проверить реализацией методы, которые позволяли бы (без принципиальных изменений) извлекать 3D модели и текстуры из любых приложений, использующих Direct3D.

Требования и ограничения: В силу жесткого ограничения по времени исследования, реализация разработанных методов должна позволять:

1. Извлекать данные из приложений Direct3D 8, поставляемых с SDK (dx8sdk\samples\Multimedia\Direct3D\bin), в силу того, что их исходный код легко доступен каждому;
2. Работать с приложениями не использующие шейдеры (для простоты реализации);
3. Сохранять: текущий установленный материал, значения вершинного буфера из потока 0, текущие значения буфера индексов и текстуру с индексом 0.

2. Идея:

Идея состоит в подмене функций COM объектов Direct3D, из которых доступны необходимые для извлечения данные.

Все рассматриваемые приложения (1.1), используют для создания COM объекта Direct3D8 функцию Direct3DCreate8 из d3d8.dll. Эта функция возвращает адрес указателя на таблицу указателей на функции объекта Direct3D8 (согласно интерфейсу IDirect3D8). Необходимо ее подменить. При вызове функций CreateDevice из COM объекта Direct3D8 создается COM объект Direct3DDevice8 – основной объект приложения, использующего Direct3D, этот объект имеет функцию DrawIndexedPrimitive, которая используется для «отрисовки» трехмерных моделей в (1.1). Подмены функции IDirect3D8.CreateDevice, на функцию, которая выполнит действительную CreateDevice и подменит в объекте IDirect3DDevice8 DrawIndexedPrimitive, которая выполнит:

1. сохранение данных в файлы mesh.x и tex.bmp;

2. вызов действительной функции DrawIndexedPrimitive;
3. возврат к выполнению программы.

вполне достаточно для реализации цели.

Вся необходимая исходная информация получена из [1], [2], [3], [4], [5].

3. Алгоритмы

1. Подмена библиотеки d3d8.dll естественно позволяет подменить функцию Direct3DCreate8. Для этого необходимо создать соответствующую библиотеку dll, экспортирующую Direct3DCreate8 и скопировать ее в директорию целевого приложения [6].
2. Подмена функций COM объектов производится записью в таблицу адресов реализованных согласно (2) функций, с сохранением адресов действительных функций.
3. Параметр obj объектно-ориентированных подмененных функций позволяет получить доступ ко всем функциям COM объекта, и выполнить сохранение необходимых данных.

4. Реализация

Реализация произведена в Flat assembler – version 1.62 by Tomasz Grysztar. При реализации проверка ошибок вызовов сознательно игнорируется, как не входящая в рамки рассматриваемой работы (целью которой не является создание приложения сохраняющего модели из любых программ).

1. Создание dll базируется на примере, поставляемом с fasm. Приведем реализацию функции Direct3DCreate8:

```
; реализация экспортируемой функции Direct3DCreate8.
d3d8lib  db 'c:\windows\system32\d3d8.dll',0  ; путь к «дейсвительной» библиотеке
d3d8proc db 'Direct3DCreate8',0    ; имя «действительной» фунции

proc MyDirect3DCreate8 SDKVersion
 invoke  LoadLibrary,d3d8lib   ; загрузка d3d8.dll
 invoke  GetProcAddress,eax,d3d8proc  ; получение адреса Direct3DCreate8
 stdcall eax,[SDKVersion]   ; выполнение «действительной» Direct3DCreate8

 ; сохранение адреса IDirect3D8.CreateDevice в переменную CreateDevice и замещение его на адрес
    MyCreateDevice
 stdcall replace,[eax],IDirect3D8_CreateDevice,MyCreateDevice,CreateDevice
 ret
endp
```

2. Функция замещения адресов в таблице адресов функций COM объекта имеет следующие параметры: IntTblAddr – адрес таблицы, MethodNumber – номер функции, NewFunction — адрес замещающей функции, SaveOldFunction – адрес переменной для сохранения адреса исходной функции.

```
; реализация функции замещения адресов в COM интерфейсах
proc replace IntTblAddr,MethodNumber,NewFunction,SaveOldFunction
 push eax
 push ebx

 ; вычисление адреса элемента инерфейса MethodNumber*4+IntTblAddr.
 mov eax,[MethodNumber]
 shl eax,2
 add eax,[IntTblAddr]

 ; проверка: не заменена ли уже функция.
```

```
 mov ebx,[eax]
 cmp [NewFunction],ebx
 je AlreadyReplaced

 ; сохранение адреса исходной функции
 push eax
 mov eax,[SaveOldFunction]
 mov [eax],ebx

 ; запись адреса замещающей функции
 mov eax,[esp]
 invoke VirtualQuery,eax,mbi,MEMORY_BASIC_INFORMATIONsz

 mov eax,[esp]
 push [mbi.Protect]
 invoke VirtualProtect,eax,4,PAGE_EXECUTE_READWRITE,esp
 add esp,4

 pop eax

 mov ebx,[NewFunction]
 mov [eax],ebx ; непосредственно запись

 push PAGE_EXECUTE_READWRITE
 invoke VirtualProtect,eax,4,[mbi.Protect],esp
 add esp,4

AlreadyReplaced:

 pop ebx
 pop eax
 ret
endp
```

3. Реализация функций MyCreateDevice и MyDrawIndexedPrimitive является типовой для всех замещаемых функций. В MyCreateDevice вызывается CreateDevice и замещается DrawIndexedPrimitive. В MyDrawIndexedPrimitive, если нажата клавиша F12 происходит сохранение данных, далее вызывается DrawIndexedPrimitive. Параметры функций соответствуют параметрам замещенных [6]:

```
; реализация замещающих функций
proc MyCreateDevice obj,Adapter,DeviceType,hFocusWindow,BehaviorFlags,pPresentationParameters,
    ppReturnedDeviceInterface
 ; Вызов исходной функции IDirect3D8.CreateDevice
 invoke CreateDevice,[obj],[Adapter],[DeviceType],[hFocusWindow],[BehaviorFlags],[
    pPresentationParameters],[ppReturnedDeviceInterface]

 ; получение адреса таблицы адресов функций объекта Direct3DDevice8
 push eax
 mov eax,[ppReturnedDeviceInterface]
 mov eax,[eax]

 ; замена DrawIndexedPrimitive
 stdcall replace,[eax],IDirect3DDevice8_DrawIndexedPrimitive,MyDrawIndexedPrimitive,
    DrawIndexedPrimitive
 pop eax
 ret
endp

proc MyDrawIndexedPrimitive obj,Type,MinIndex,NumVertices,StartIndex,PrimitiveCount
 ; проверка нажатия F12
 invoke GetAsyncKeyState,VK_F12
 test eax,eax
 jns donotsave

 ; сохранение модели
 stdcall Save, [obj]

donotsave:
 ; вызов исходной DrawIndexedPrimitive
```

```
  invoke DrawIndexedPrimitive,[obj],[Type],[MinIndex],[NumVertices],[StartIndex],[PrimitiveCount]
  ret
endp
```

4. Реализация функции сохранения текстуры и трехмерного объекта не представляет особого интереса с точки зрения текущего исследования, приведем ее вкратце:

1. У объекта obj получим установленную текстуру IDirect3DDevice8.GetTexture;
2. Далее сохраним текстуру с помощью D3DXSaveTextureToFile;
3. Получим текущий материал IDirect3DDevice8.GetMaterial;
4. Установим в материале имя файла текущей текстуры;
5. Получим объект индексов Direct3DIndexBuffer8 через IDirect3DDevice8.GetIndices;
6. Определим формат индексов и размер буфера индексов IDirect3DIndexBuffer8.GetDesc;
7. Посчитаем количество треугольников в 3D объекте: (размер буфера индексов)/(размер одного индекса)/3;
8. Получим объект Direct3DVertexBuffer8, хранящий данные вершин: IDirect3DDevice8.GetStreamSource;
9. Получим описание размера буфера вершин: IDirect3DVertexBuffer8.GetDesc;
10. Получим размер одной вершины: D3DXGetFVFVertexSize;
11. Посчитаем количество всех вершин: (размер буфера)/(размер одной вершины);
12. Создадим объект D3DXMesh для сохранения данных в файл: D3DXCreateMeshFVF;
13. Откроем буфер вершин obj для чтения, а D3DXMesh – для записи: IDirect3DVertexBuffer8.Lock, ID3DXMesh.LockVertexBuffer;
14. Запишем данные в буфер объекта D3DXMesh;
15. Закроем буферы: IDirect3DIndexBuffer8.Lock, ID3DXMesh.LockIndexBuffer;
16. Выполним шаги 12 – 14 для индексов;
17. Сохраним данные в файл формата DirectX: D3DXSaveMeshToX.

5. Результаты

Созданная программа (несмотря на существенные ограничения ее применимости) позволяет доказать реализуемость рассматриваемой технологии.

6. Возможное применение

Наиболее эффективные, на мой взгляд, способы применения описанной технологии в приложениях Direct3D это:

1. Модификация трехмерных объектов приложений и их текстур;
2. Извлечение моделей из приложений и использование их в собственных целях (если это не нарушает авторских прав создателей);
3. При замещении всех функций интерфейсов Direct3D возможна реализация «трехмерных» копий сцен приложений.

7. Достаточные источники:

1. Эш Рофэйл, Яссер Шохауд. COM и COM+. Полное руководство. Пер. с англ. – К.: ВЕК +, К.: НТИ, М.: Энтроп, 2000.

2. April 2003 Release of the MSDN Library.
3. d3d8.h, входящий в состав Microsoft Development Environment 2003.
4. d3dfile.cpp из DirectX8 SDK.
5. DirectX 8.1 Programmer's Reference.
6. Рихтер Дж. Windows для профессионалов: создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows/Пер. с англ. – 4-у изд. — СПб: Питер; М.: Издательско-торговый дом «Русская редакция», 2001.