# Разработка COM-компонента

***Задание*:** Разработать ко-класс, моделирующий функционирование виртуального автомобиля. Список функций, которые должна будет поддерживать программа:

1. Позволять пользователю создавать машину, присваивая ей имя и определяя максимальную скорость.
2. Обеспечивать, чтобы максимальная скорость не превышала 500 км/час.
3. При необходимости отображать на экране максимальную скорость и имя машины.
4. Ускорять машину с приращением в 10 км/час.
5. Определять, не треснул ли двигатель из-за чрезмерной скорости (срабатывает при первом задании приращения в 10 км/час сверх максимальной скорости).

Класс должен поддерживать три пользовательских интерфейса.

## План разработки компонента

Для выполнения задания нужно выполнить следующие шаги:

1. Разработать класс *СоСаr* и реализовать в нем все необходимые интерфейсы.
2. Разработать фабрику класса *СоСаr* для его активизации по запросу клиента.
3. Реализовать экспортные функции *DllGetClassObject()* и *DllCanUnloadNow()*.
4. Создать регистрационный файл для внесения в реестр Windows информации о разработанном компоненте.
5. Проектирование интерфейса

Запустите среду разработки Microsoft Visual Studio и создайте новый проект С++ (тип проекта библиотека динамической компоновки DLL), в окне мастера **Win32 Application Wizard** установите параметр **Empty application** (рис. 7).

Добавьте в проект заголовочный файл с именем *interfaces.h*, в котором определите три интерфейса, моделирующие поведение автомобиля.

#ifndef \_INTERFACES

#define \_INTERFACES

#include <windows.h>

//IStats используется для получения имени автомобиля, а также

//для отображения параметров состояния реализуемого объекта

DECLARE\_INTERFACE\_(IStats, IUnknown)

{

STDMETHOD(DisplayStats)() PURE;

STDMETHOD(GetPetName)(BSTR\* petName) PURE;

};

//IEngine моделирует поведение двигателя. Можем ускорять его,

//получать значения максимальной и текущей скорости

DECLARE\_INTERFACE\_(IEngine, IUnknown)

{

STDMETHOD(SpeedUp)() PURE;

STDMETHOD(GetMaxSpeed)(int\* maxSpeed) PURE;

STDMETHOD(GetCurSpeed)(int\* curSpeed) PURE;

};

//ICreateCar используется для присвоения имени реализуемому

//объекту и задания максимальной скорости

DECLARE\_INTERFACE\_(ICreateCar, IUnknown)

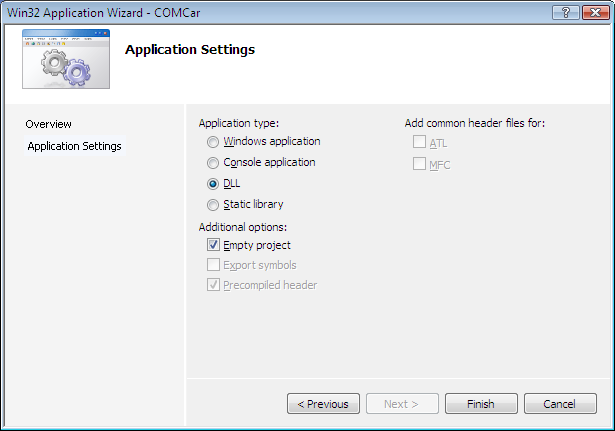
{

STDMETHOD(SetPetName)(BSTR petName) PURE;

STDMETHOD(SetMaxSpeed)(int maxSp) PURE;

};

#endif // \_INTERFACES



1. **Настройка параметров проекта**

Все пользовательские интерфейсы СОМ должны быть произведены от интерфейса *IUnknown*, поэтому необходимо подключить заголовочный файл *windows.h*, в котором объявлен требуемый интерфейс. Директивы препроцессора *ifndef*, *define*, *endif* используются для предотвращения ошибки по­вторного определения.

После определения интерфейсов каждому из них необходимо присвоить GUID (Globally Unique Identifier). Для этого создайте еще один заголовочный файл с именем *iid.h*, который будет содержать описания макросов *DEFINE\_GUID*. Используя утилиту *guidgen.exe* или команду среды разработки **Tools | Create CUID**, создайте три новых GUID с соответствующими константами и префиксом *IID\_*. Ниже представлены возможные значения GUID (очевидно, что полученные Вами значения будут отличаться).

// {A533DA30-D372-11d2-B8CF-0020781238D4}

DEFINE\_GUID(IID\_IEngine,

0xa533da30, 0xd372, 0x11d2, 0xb8, 0xcf, 0x0, 0x20, 0x78, 0x12, 0x38, 0xd4);

// {A533DA31-D372-11d2-B8CF-0020781238D4}

DEFINE\_GUID(IID\_IStats,

0xa533da31, 0xd372, 0x11d2, 0xb8, 0xcf, 0x0, 0x20, 0x78, 0x12, 0x38, 0xd4);

// {A533DA32-D372-11d2-B8CF-0020781238D4}

DEFINE\_GUID(IID\_ICreateCar,

0xa533da32, 0xd372, 0x11d2, 0xb8, 0xcf, 0x0, 0x20, 0x78, 0x12, 0x38, 0xd4);

Создайте файл *iid.cpp* (**Project|Add To Project|Files**) и добавьте в него приведенные ниже директивы.

#include <windows.h>

#include <initguid.h>

#include "iid.h"

Заголовочный файл *windows.h* используется для объявления констант и структур, относящихся к COM, *initguid.h* для определения *DEFINE\_GUID*. Централизованный файл для всех GUID может быть очень полезен, особенно если вам потребуется включать определения в другие проекты. Преимущество иметь соответствующий файл \*.срр для ID ваших интерфейсов состоит в том, чтобы гарантировать автоматический вызов *<initguid.h>* перед вашим заголовочным файлом, а не полагаться в этом на конечного пользователя ваших IID.

1. Реализация IUnknown

Добавьте в проект новый класс C++ с именем *CoCar*, содержащий интерфейсы *IEngine*, *IStats* и *ICreateCar*. Поскольку каждый интерфейс СОМ производится от *IUnknown*, вам понадобится реализовать не только методы каждого пользовательского интерфейса, но и три метода IUnknown.

#include "interfaces.h"

class CoCar : public IEngine, public ICreateCar, public IStats

{

public:

CoCar();

virtual ~CoCar();

// IUnknown

STDMETHODIMP QueryInterface(REFIID riid, void\*\* pIFace);

STDMETHODIMP\_(DWORD)AddRef();

STDMETHODIMP\_(DWORD)Release();

// IEngine

STDMETHODIMP SpeedUp();

STDMETHODIMP GetMaxSpeed(int\* maxSpeed);

STDMETHODIMP GetCurSpeed(int\* curSpeed);

// IStats

STDMETHODIMP DisplayStats();

STDMETHODIMP GetPetName(BSTR\* petName);

// ICreateCar

STDMETHODIMP SetPetName(BSTR petName);

STDMETHODIMP SetMaxSpeed(int maxSp);

};

Реализацию *IUnknown* начнем со счетчика ссылок. Добавьте к классу *CoCar* private переменную m\_*refCount* типа *ulong*, которая будет являться счетчиком ссылок, и обнулите ее в конструкторе. Функция *AddRef()* будет добавлять к счетчику единицы, а *Release()* – отнимать единицы и проверять на оконча­тель­ный сброс, после которого объект сам себя выгрузит из памяти.

STDMETHODIMP\_(DWORD) CoCar::AddRef()

{

++m\_refCount;

return m\_refCount;

}

STDMETHODIMP\_(DWORD) CoCar::Release()

{

if(--m\_refCount == 0)

{

delete this;

return 0;

}

else

return m\_refCount;

}

Необходимо также реализовать функцию *QueryInterface()*, которая будет возвращать указатели интерфейса для *IUnknown*, *IStats*, *ICreateCar* и *IEngine*. Логика работы данной функции довольно проста. Если функция предоставила интерфейс, то необходимо вызвать *AddRef()* (добавляется еще один клиент) и вернуть ссылку на данный объект, преобразовав его к требуемому интерфейсу, в случае если запрашиваемый интерфейс не поддерживается, необходимо вернуть константу *E\_NOINTERFACE*.

STDMETHODIMP CoCar::QueryInterface(REFIID riid, void\*\* pIFace)

{

// Which aspect of me do they want?

if(riid == IID\_IUnknown)

{

\*pIFace = (IUnknown\*)(IEngine\*)this;

MessageBox(NULL, "Handed out IUnknown","QI",MB\_OK |   
 MB\_SETFOREGROUND);

}

else if(riid == IID\_IEngine)

{

\*pIFace = (IEngine\*)this;

MessageBox(NULL, "Handed out IEngine","QI",MB\_OK |   
 MB\_SETFOREGROUND);

}

else if(riid == IID\_IStats)

{

\*pIFace = (IStats\*)this;

MessageBox(NULL, "Handed out IStats","QI",MB\_OK |   
 MB\_SETFOREGROUND);

}

else if(riid == IID\_ICreateCar)

{

\*pIFace = (ICreateCar\*)this;

MessageBox(NULL, "Handed out ICreateCar","QI",MB\_OK |   
 MB\_SETFOREGROUND);

}

else

{

\*pIFace = NULL;

return E\_NOINTERFACE;

}

((IUnknown\*)(\*pIFace))->AddRef();

return S\_OK;

}

1. Реализация методов пользовательских интерфейсов

После реализации *IUnknown* необходимо реализовать пользовательские интерфейсы *CoCar*. Обратите внимание на использование функции работы со строками при работе с типом *BSTR*. Добавьте в класс следующие данные, затем инициализируйте их в конструкторе.

BSTR m\_petName; // Инициализация через SysAllocString(),   
 // удаление — через вызов SysFreeString()

int m\_maxSpeed; // Максимальная скорость

int m\_currSpeed; // Текущая скорость СоСаr

// Конструктор и деструктор СоСаr

CoCar::CoCar() : m\_refCount(0), m\_currSpeed(0), m\_maxSpeed(0)

{

m\_petName = SysAllocString(L"Default Pet Name");

}

CoCar::~CoCar()

{

if(m\_petName)

SysFreeString(m\_petName);

MessageBox(NULL, "CoCar is dead","Destructor",MB\_OK |   
 MB\_SETFOREGROUND);

}

Реализация интерфейса *IEngine* приведена ниже. Метод *SpeedUp()* увеличивает значение *m\_currSpeed* на 10 и возвращает *S\_OK*, *GetMaxSpeed()* позволяет получить значение пере­менной *m\_maxSpeed*, a *GetCurrSpeed()* - значение m\_currSpeed.

// Реализация IEngine

STDMETHODIMP CoCar::SpeedUp()

{

m\_currSpeed += 10;

return S\_OK;

}

STDMETHODIMP CoCar::GetMaxSpeed(int\* maxSpeed)

{

\*maxSpeed = m\_maxSpeed;

return S\_OK;

}

STDMETHODIMP CoCar::GetCurSpeed(int\* curSpeed)

{

\*curSpeed = m\_currSpeed;

return S\_OK;

}

Интерфейс *ICreateCar* имеет два метода. *SetPetName()* принимает заданную клиентом *BSTR* и помещает ее во внутренний буфер *BSTR*, используя функцию *SysReAllocString()*. *SetMaxSpeed()* устанавливает максимальную скорость, следя при этом, чтобы она не превышала константу *MAX\_SPEED*, которую вы должны задать в заголовочном файле класса.

// Реализация ICreateCar

STDMETHODIMP CoCar::SetPetName(BSTR petName)

{

SysReAllocString(&m\_petName, petName);

return S\_OK;

}

STDMETHODIMP CoCar::SetMaxSpeed(int maxSp)

{

if(maxSp < MAX\_SPEED)

m\_maxSpeed = maxSp;

return S\_OK;

}

Реализуйте функции интерфейса *IStats*. Метод *GetPetName()* возвращает копию внутреннего буфера, используя функцию *SysAllocString()*.

// Возвращает клиенту копию внутреннего буфера BSTR

STDMETHODIMP CoCar::GetPetName(BSTR\* petName)

{

\*petName = SysAllocString(m\_petName);

return S\_OK;

}

Метод *DisplayStats()* просто предоставляет два блока сообщений для отображения текущей скорости и имени машины, преобразовывая Unicode-строку в ANSI.

// Информация о СоСаr помещается в блоки сообщений

STDMETHODIMP CoCar::DisplayStats()

{

// Need to transfer a BSTR to a char array.

char buff[MAX\_NAME\_LENGTH];

WideCharToMultiByte(CP\_ACP, NULL, m\_petName, -1, buff,

MAX\_NAME\_LENGTH, NULL, NULL);

MessageBox(NULL, buff, "Pet Name",MB\_OK | MB\_SETFOREGROUND);

memset(buff, 0, sizeof(buff));

sprintf(buff, "%d", m\_maxSpeed);

MessageBox(NULL, buff, "Max Speed", MB\_OK|   
 MB\_SETFOREGROUND);

return S\_OK;

}

Попробуйте откомпилировать проект и убедитесь, что синтаксические ошибки отсутствуют.

Добавьте новый файл с именем *CarlnProcServer.cpp*. Функции и параметры, которые мы внесем в этот файл, образуют компоненты нашей DLL. Для начала объявите две глобальных переменных типа *ULONG* для счетчика блокировок и счетчика объектов. Убедитесь, что новый файл вставлен в рабочее пространство проекта:

ULONG g\_lockCount = 0; // Количество блокировок сервера

ULONG g\_objCount = 0; // Количество "живых" объектов в сервере

Поскольку данная версия *CoCar* будет размещена в двоичном файле, dfv потребуется *CLSID* для уникальной идентификации *CoCar* в среде СОМ. Сгенерируйте новый *GUID* (с помощью guidgen.exe), который станет *CLSID* для *CoCar*. Определите константу *CLSID\_CoCar* в качестве первого параметра макроса *DEFINE\_GUID*. Добавьте этот GUID в существующий файл iid.h:

// {7AD2D539-EE35-11d2-B8DE-0020781238D4}

DEFINE\_GUID(CLSID\_CoCar,

0x7ad2d539, 0xee35, 0x11d2, 0xb8, 0xde, 0x0, 0x20, 0x78, 0x12, 0x38, 0xd4);

Также нужно реализовать увеличение и уменьшение глобального счетчика объектов в конструкторе и деструкторе класса *CoCar*. Вспомните, что мы должны предоставить «точки захвата», чтобы информировать DLL-сервер о том, сколько объектов на данный момент «живут» в нем. Используйте ключевое слово *extern* для обращения к глобальной переменной *g\_objCount* внутри класса CoCar:

extern DWORD g\_objCount;

CoCar::CoCar() : m\_refCount(0), m\_currSpeed(0), m\_maxSpeed(0)

{

// Прочий код

++g\_objCount;

}

CoCar::~CoCar()

{

--g\_objCount;

// Прочий код

}

1. Разработка фабрики класса   
   для CoCar

Для создания *CoCar* нам понадобится фабрика класса. Вставьте новый класс с именем *CoCarClassFactory*, производный от *IClassFactory*. Вспомните, что объект класса обеспечивает языково-независимый способ создания объектов СОМ. Вот начальное определение нашего объекта класса:

#include <windows.h>

class CoCarClassFactory : public IClassFactory

{

public:

CoCarClassFactory();

virtual ~CoCarClassFactory();

// IUnknown

STDMETHODIMP QueryInterface(REFIID riid, void\*\* pIFace);

STDMETHODIMP\_(DWORD)AddRef();

STDMETHODIMP\_(DWORD)Release();

// ICF

STDMETHODIMP LockServer(BOOL fLock);

STDMETHODIMP CreateInstance(LPUNKNOWN pUnkOuter,   
 REFIID riid, void\*\* ppv);

private:

DWORD m\_refCount;

};

Начнем с реализации методов *IUnknown*, *AddRef()*, *Release()* и *QueryInterface()*. Код почти полностью совпадает с реализацией *IUnknown* для *CoCar*, за исключением того, что на этот раз *QueryInterfасе()* проверяет только *IID\_IUnknown* и *IID\_IClassFactory*:

STDMETHODIMP\_(ULONG) CoCarClassFactory::AddRef()

{

return ++m\_refCount;

}

STDMETHODIMP\_(ULONG) CoCarClassFactory::Release()

{

if(--m\_refCount == 0)

{

delete this;

return 0;

}

return m\_refCount;

}

STDMETHODIMP CoCarClassFactory::QueryInterface(REFIID riid, void\*\* ppv)

{

// Which aspect of me do they want?

if(riid == IID\_IUnknown)

{

\*ppv = (IUnknown\*)this;

}

else if(riid == IID\_IClassFactory)

{

\*ppv = (IClassFactory\*)this;

}

else

{

\*ppv = NULL;

return E\_NOINTERFACE;

}

((IUnknown\*)(\*ppv))->AddRef();

return S\_OK;

}

Теперь реализуем *IClassFactory::CreateInstance()*. В этом методе создадим объект *CoCar* и запросим нужный интерфейс. Вернем *HRESULT*, который клиент сможет проанализировать. Если в методе *CreateInstance()* что-то не сработает, выделенную под ко-класс память надо освободить:

STDMETHODIMP CoCarClassFactory::CreateInstance(LPUNKNOWN pUnkOuter, REFIID riid, void\*\* ppv)

{

if(pUnkOuter != NULL)

return CLASS\_E\_NOAGGREGATION;

CoCar\* pCarObj = NULL;

HRESULT hr;

pCarObj = new CoCar;

hr = pCarObj -> QueryInterface(riid, ppv);

if (FAILED(hr))

delete pCarObj;

return hr; // S\_OK or E\_NOINTERFACE.

}

Чтобы закончить с фабрикой класса, реализуем *IClassFactory::LockServer()* для работы со счетчиком блокировок. В конструкторе и деструкторе объекта класса добавьте соответствующий код для этого счетчика. Для обращения к глобальным переменным, определенным в *CarInProcServer.cpp*, используйте ключевое слово *extern* в файле СРР вашего объекта класса:

extern DWORD g\_lockCount;

extern DWORD g\_objCount;

CoCarClassFactory::CoCarClassFactory()

{

m\_refCount = 0;

g\_objCount++;

}

CoCarClassFactory::~CoCarClassFactory()

{

g\_objCount--;

}

STDMETHODIMP CoCarClassFactory::LockServer(BOOL fLock)

{

if(fLock)

++g\_lockCount;

else

--g\_lockCount;

return S\_OK;

}

Кроме того, убедитесь, что вы вставили нужные файлы в исходный код вашей фабрики класса. Скомпилируйте и убедитесь в отсутствии синтаксических ошибок. Следующим шагом будет разработка минимального полного набора экспортируемых функций сервера.

1. Реализация хранилища   
   компонентов DLL

Реализуйте в файле *CarInProcServer.cpp* метод *DllGetClassObject()* для выполнения следующих операций:

STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid, void\*\* ppv)

{

HRESULT hr;

CoCarClassFactory \*pCFact = NULL;

if(rclsid != CLSID\_CoCar)

return CLASS\_E\_CLASSNOTAVAILABLE;

pCFact = new CoCarClassFactory;

hr = pCFact -> QueryInterface(riid, ppv);

if(FAILED(hr))

delete pCFact;

return hr;

}

Последняя экспортируемая функция DLL, которую мы реализуем – это *DllCanUnloadNow()*:

STDAPI DllCanUnloadNow()

{

if(g\_lockCount == 0 && g\_objCount == 0)

{

return S\_OK;

}

else

return S\_FALSE;

}

Теперь создайте и вставьте в ваш проект новый файл *carinprocserver.def*, чтобы сделать эти две функции видимыми окружающему миру. Необходимо, чтобы имя в операторе *LIBRARY* совпадало с именем проекта, который вы компонуете:

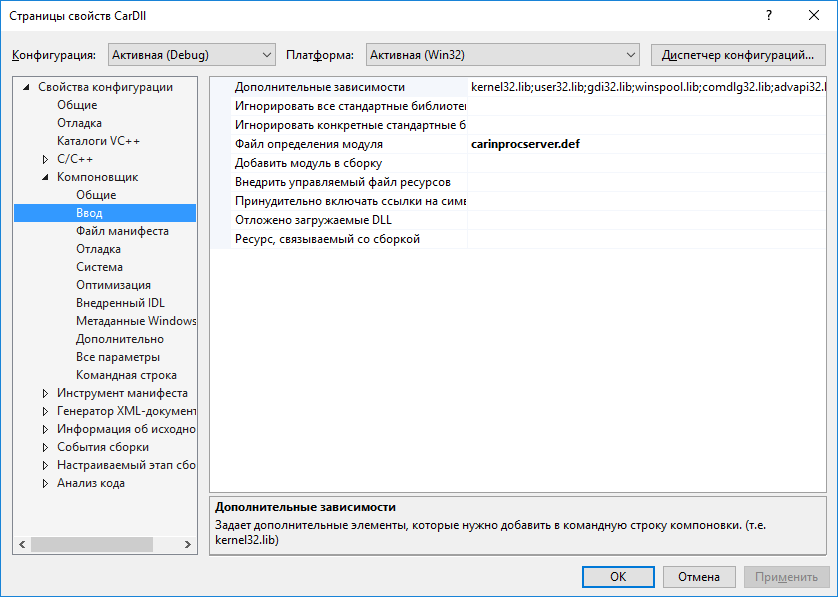
LIBRARY "CARINPROCSERVER"

EXPORTS

DllGetClassObject @1 PRIVATE

DllCanUnloadNow @2 PRIVATE

В окне свойств проекта убедитесь, что добавленный файл появился в свойстве «Файл определения модуля». При его отсутствии введите значение вручную.



1. Внесение информации в реестр

Чтобы завершить наш DLL-сервер, нам нужно создать еще REG-файл для внесения минимальной и полной информации касательно сервера в реестр системы. Создайте *ProgID* для вашего ко-класса с подключом CLSID. Введите его прямо под *HKCR*. Затем введите *CLSID* для вашего ко-класса с подключом *InprocServer32*, указывающим на физический путь к DLL.

Помните, что между символами слэш (\) не должно быть пробелов, а оператор присваивания (=), наоборот, должен быть взят в пробелы. Также смотрите, чтобы в первой строке файла стоял тэг REGEDIT. Вот возможный вариант файла *carinprocserver.reg* (разумеется, вы должны будете подставить ваши *CLSID* и путь):

REGEDIT

HKEY\_CLASSES\_ROOT\CarInProcServer.CoCar\CLSID =   
 {7AD2D539-EE35-11d2-B8DE-0020781238D4}

HKEY\_CLASSES\_ROOT\CLSID\{7AD2D539-EE35-11d2-B8DE-0020781238D4} =   
 CarInProcServer.CoCar

HKEY\_CLASSES\_ROOT\CLSID\{7AD2D539-EE35-11d2-B8DE-0020781238D4}\InprocServer32 = Путь до компонента\CarInProcServer.dll

Для 64-х разрядных версий Windows необходимо доплнительно зарегистрировать класс в разделе: HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\Classes

1. Разработка COM-клиента на С++

Создайте новое консольное приложение с именем *CoCarClient*. Добавьте в проект файлы: *iid.h*, *iid.cpp*, *interfaces.h*. Ниже представлен полный программный код клиента.

#include "interfaces.h"

#include "iid.h"

#include <iostream.h>

int main()

{

CoInitialize(NULL); // This parameter is reserved,   
 // and should be NULL

HRESULT hr;

IClassFactory\* pCF = NULL;

ICreateCar\* pICreateCar = NULL;

IStats\* pStats = NULL;

IEngine\* pEngine = NULL;

hr = CoGetClassObject(CLSID\_CoCar, CLSCTX\_INPROC\_SERVER,

NULL, IID\_IClassFactory, (void\*\*)&pCF);

// Make a CoCar & get ICreateCar.

hr = pCF->CreateInstance(NULL, IID\_ICreateCar,   
 (void\*\*)&pICreateCar);

pCF->Release();

if(SUCCEEDED(hr))

{

pICreateCar->SetMaxSpeed(30);

BSTR petName = SysAllocString(L"Shazzam!");

pICreateCar->SetPetName(petName);

SysFreeString(petName);

// Now get IStats

hr = pICreateCar->QueryInterface(IID\_IStats,   
 (void\*\*)&pStats);

pICreateCar->Release();

}

if(SUCCEEDED(hr))

{

// Show my car!

pStats->DisplayStats();

hr = pStats->QueryInterface(IID\_IEngine,   
 (void\*\*)&pEngine);

}

if(SUCCEEDED(hr))

{

int curSp = 0;

int maxSp = 0;

pEngine->GetMaxSpeed(&maxSp);

do // Zoom!

{

pEngine->SpeedUp();

pEngine->GetCurSpeed(&curSp);

cout << "Speed is: " << curSp << endl;

}while(curSp <= maxSp);

// Gotta convert to char array.

char buff[80];

BSTR bstr;

pStats->GetPetName(&bstr);

WideCharToMultiByte(CP\_ACP, NULL, bstr, -1, buff,

80, NULL, NULL);

cout << buff << " has blown up! Lead Foot!" << endl << endl;

SysFreeString(bstr);

if(pEngine) pEngine->Release();

if(pStats) pStats->Release();

}

CoUninitialize();

return 0;

}

1. Разработка COM-клиента на С#

Опишите заглушки для класса и интерфейсов.

[ComVisible(true)]

[ComImport, Guid("754CD3AC-DC52-4256-9A90-B03FB283FF56")]

public class Car

{

}

//{960EFB55-5A90-45F8-8C6C-12849DCF2AC1}

//DEFINE\_GUID(IID\_ICreateCar,

//0x960efb55, 0x5a90, 0x45f8, 0x8c, 0x6c, 0x12, 0x84, 0x9d, 0xcf, 0x2a, 0xc1);

[ComImport, InterfaceType(ComInterfaceType.InterfaceIsIUnknown), Guid("960EFB55-5A90-45F8-8C6C-12849DCF2AC1")]

public interface ICreateCar

{

void SetPetName(string petName);

void SetMaxSpeed(int maxSp);

}

// {4601678B-6D77-446F-8782-CC054440F81F}

//DEFINE\_GUID(IID\_IStats,

//0x4601678b, 0x6d77, 0x446f, 0x87, 0x82, 0xcc, 0x5, 0x44, 0x40, 0xf8, 0x1f);

[ComImport, InterfaceType(ComInterfaceType.InterfaceIsIUnknown), Guid("4601678B-6D77-446F-8782-CC054440F81F")]

public interface IStats

{

void DisplayStats();

void GetPetName( ref string petName);

}

[ComImport, InterfaceType(ComInterfaceType.InterfaceIsIUnknown), Guid("0010D07E-8AF4-4436-A4D1-A2F57C6A4DA4")]

public interface IEngine

{

void SpeedUp();

void GetMaxSpeed(ref int curSpeed);

void GetCurSpeed(ref int maxSpeed);

}

Использование компонента:

Car myCar = new Car();

ICreateCar iCrCar = (ICreateCar)myCar;

Console.WriteLine("Напишите имя: ");

iCrCar.SetPetName(Console.ReadLine());

1. Реализация интерфейса IDispatch

Реализуйте разработанному компоненту интерфейс IDispatch. (Самостоятельно)

1. Шаг 8. Разработка COM-клиента через интерфейс IDispatch

Реализуйте вызов компонента на С# с использованием интерфейса IDispatch.