Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Системное программирование

Студент: Лопатнюк П.В.

ФИТ 3 курс 1 группа

Преподаватель: Бернацкий П.В.

Минск 2024

**Лабораторная работа №4**

**Component Object Model**

**Цель работы**: Получение практических навыков в работе с Component Object Model.

**Постановка задачи:**

Взять готовый проект с сайта «diskstation.belstu.by:5001» по пути «Для\_студентов\_ФИТ\_БГТУ/ПРЕПОДАВАТЕЛИ/Бернацкий/Системное программирование - 3 курс ИСиТ/Лекции/Примеры для лекций/5/COM+CMake.7z».

Изучить проект и полностью разобраться со структурой и содержимым подпроектов.

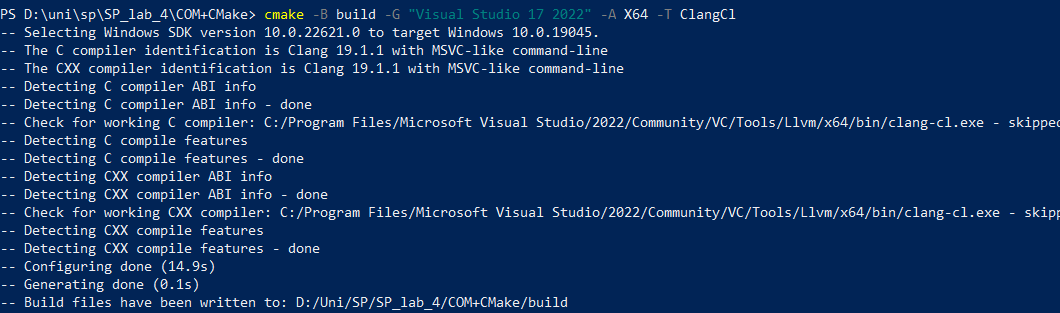
Изменить код всех проектов в необходимом объёме, а именно добавить новый интерфейс I<Ваши инициалы> который будет содержать вашу функцию из лабораторной работы №1. Если функция была написана без использования параметров (т. е., например, данные читались с консоли напрямую в функции), то обязательно параметризировать вашу функцию.

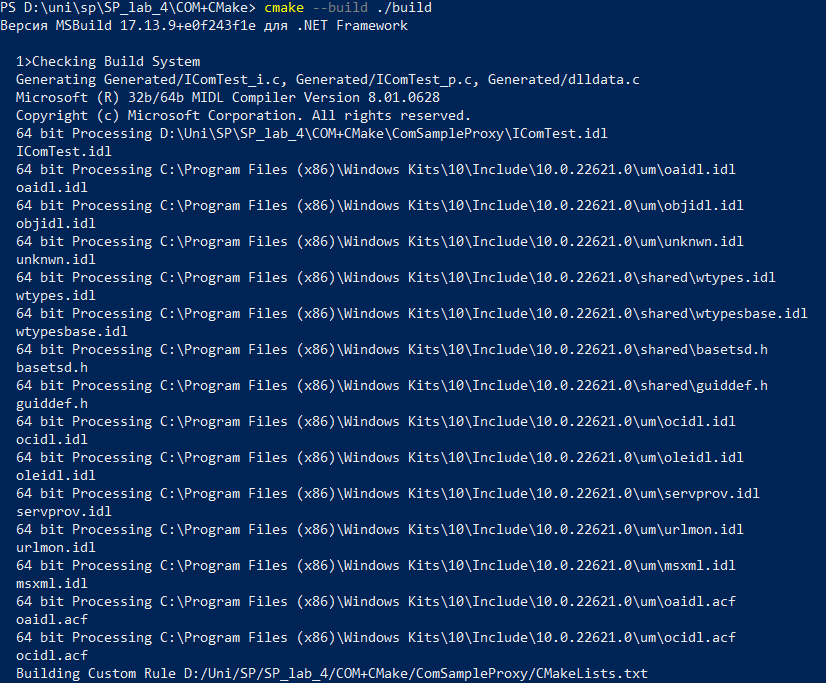
Изучить информацию из реестра о зарегистрированных COM-компонентах. (Обязательно найти оба: из проекта и ваш)

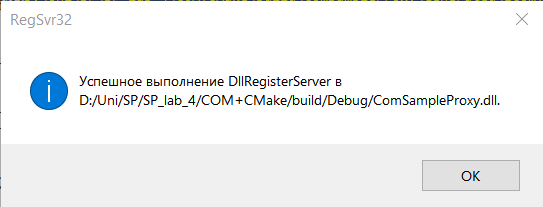
Собрать проект с помощью CMake в связке с Clang++ и продемонстрировать работоспособность получившегося проекта.

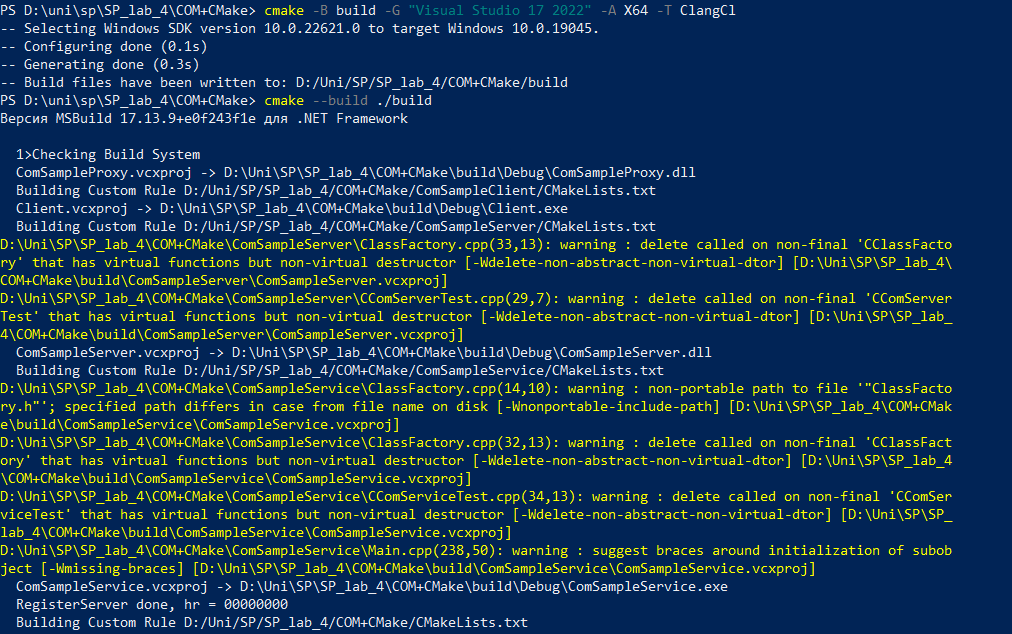
cmake -B build -G "Visual Studio 17 2022" -A X64 -T ClangCl

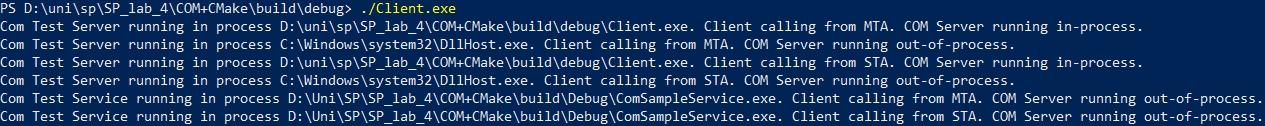
cmake --build ./build





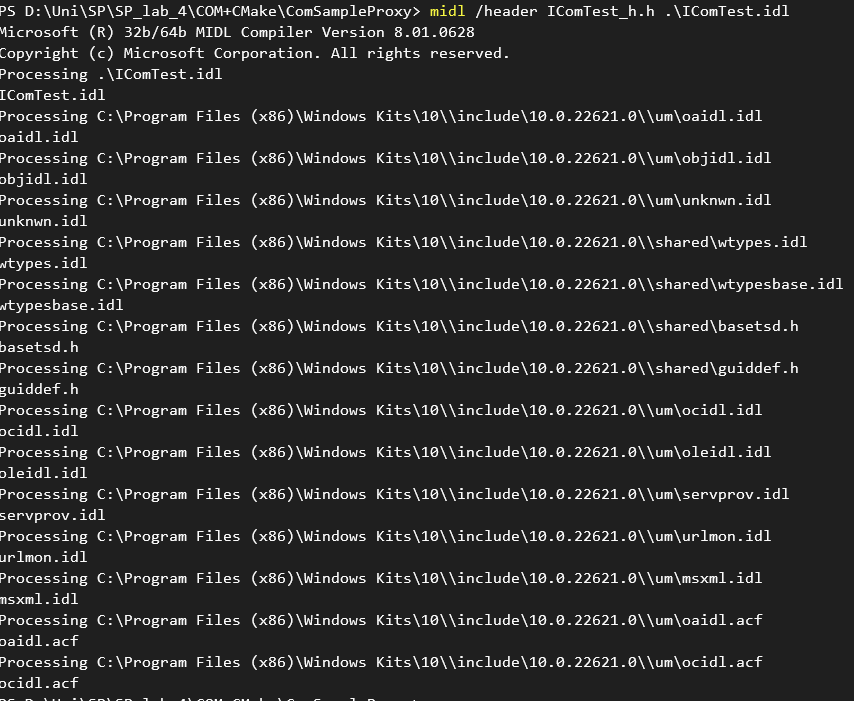


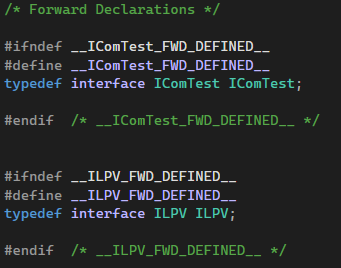


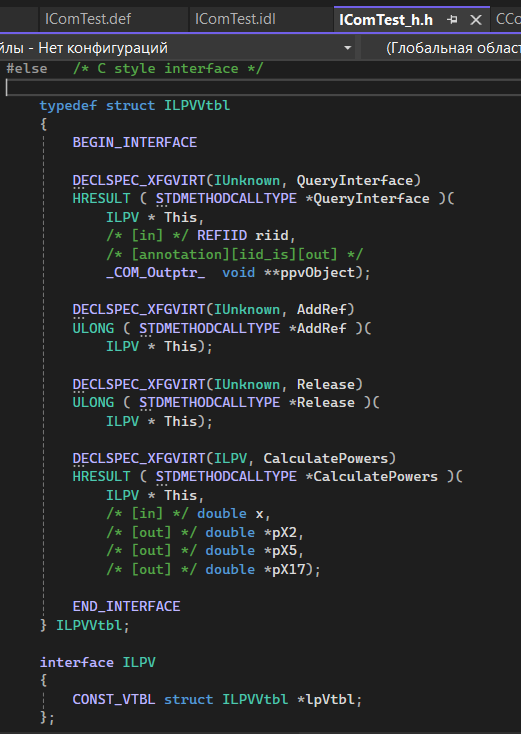
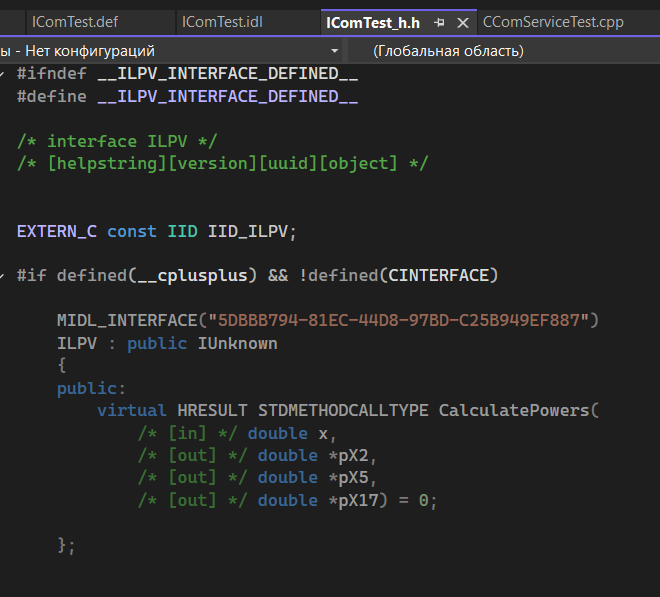


**IComTest.idl**

|  |
| --- |
| import "oaidl.idl";  import "ocidl.idl";  [  object,  uuid(C0C62619-3BC1-4095-9B9A-84503E37DAA5),  version(1.0),  helpstring("IComTest interface")  ]  interface IComTest : IUnknown  {  HRESULT WhoAmI([out] LPWSTR\* pwszWhoAmI);  };  [  object,  // Tools -> Create GUID -> Registry Format -> Copy  uuid(5DBBB794-81EC-44D8-97BD-C25B949EF887),  version(1.0),  helpstring("ILPV interface for Lab 1 calculations")  ]  interface ILPV : IUnknown  {  // Принимает 'x' на вход, возвращает результаты через указатели  HRESULT CalculatePowers(  [in] double x,  [out] double\* pX2,  [out] double\* pX5,  [out] double\* pX17  );  }; |







**CComServerTest.cpp**

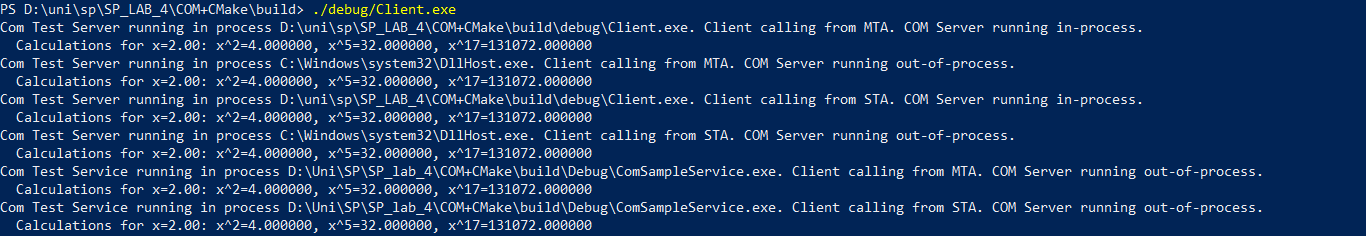
|  |
| --- |
| //  // A class that implements IComTest.  //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  #include "pch.h"  #pragma hdrstop  #include "ComSampleServerCreateInstances.h"  #include "ComSampleServerGuids.h"  #include "Dll.h"  #include <IComTest\_h.h>  #include <assert.h>  #include <objbase.h>  #include <shlwapi.h>  class CComServerTest : public IComTest, public ILPV {  public:  // IUnknown  IFACEMETHODIMP\_(ULONG) AddRef() { return InterlockedIncrement(&\_cRef); }  IFACEMETHODIMP\_(ULONG) Release() {  assert(\_cRef > 0);  ULONG cRef = InterlockedDecrement(&\_cRef);  if (0 == cRef) {  delete this;  }  return cRef;  }  IFACEMETHODIMP QueryInterface(\_\_in REFIID riid, \_\_out void \*\*ppv) {  #if 0  static const QITAB qit[] =  {  QITABENT(CComServerTest, IComTest),  { 0 },  };  return QISearch(this, qit, riid, ppv);  #endif  // ANOTHER POSSIBLE IMPLEMENTATION FOR QueryInterface METHOD.  HRESULT hr = (ppv != nullptr) ? S\_OK : E\_INVALIDARG;  if (SUCCEEDED(hr)) {  \*ppv = nullptr;  hr = E\_NOINTERFACE;  if (\_\_uuidof(IComTest) == riid) {  \*ppv = static\_cast<IComTest \*>(this);  hr = S\_OK;  } else if (\_\_uuidof(IUnknown) == riid) {  \*ppv = static\_cast<IUnknown \*>(this);  hr = S\_OK;  }  else if (riid == \_\_uuidof(ILPV)) {  \*ppv = static\_cast<ILPV\*>(this);  hr = S\_OK;  }  if (SUCCEEDED(hr)) {  reinterpret\_cast<IUnknown \*>(\*ppv)->AddRef();  }  }  return hr;  }  // IComTest  IFACEMETHODIMP WhoAmI(\_Out\_ LPWSTR \*ppwszWhoAmI) {  HRESULT hr = (ppwszWhoAmI != nullptr) ? S\_OK : E\_INVALIDARG;  if (SUCCEEDED(hr)) {  // Create a message string the contains the name of the current process.  wchar\_t wszProcessName[MAX\_PATH] = {};  DWORD dwResult = GetModuleFileNameW(nullptr, wszProcessName,  ARRAYSIZE(wszProcessName));  hr = ((dwResult < ARRAYSIZE(wszProcessName)) && (dwResult != 0))  ? S\_OK  : HRESULT\_FROM\_WIN32(GetLastError());  if (SUCCEEDED(hr)) {  LPCWSTR pwszMessagePreface = L"Com Test Server running in process ";  // Prepare the message.  size\_t cchMessagePlusTerminatingNul =  wcslen(pwszMessagePreface) + wcslen(wszProcessName) +  1 /\* For the terminating NUL character \*/;  wchar\_t \*pwszMessage = (wchar\_t \*)CoTaskMemAlloc(  cchMessagePlusTerminatingNul \* sizeof(wchar\_t));  hr = (pwszMessage != nullptr) ? S\_OK : E\_OUTOFMEMORY;  if (SUCCEEDED(hr)) {  hr = StringCchPrintfW(pwszMessage, cchMessagePlusTerminatingNul,  L"%s%s", pwszMessagePreface, wszProcessName);  if (SUCCEEDED(hr)) {  hr = SHStrDupW(pwszMessage, ppwszWhoAmI);  }  CoTaskMemFree(pwszMessage);  }  }  }  return hr;  }  IFACEMETHODIMP CalculatePowers(  /\* [in] \*/ double x,  /\* [out] \*/ double\* pX2,  /\* [out] \*/ double\* pX5,  /\* [out] \*/ double\* pX17) override  {  if (pX2 == nullptr || pX5 == nullptr || pX17 == nullptr)  {  return E\_POINTER;  }  double x2 = x \* x;  double x4 = x2 \* x2;  double x8 = x4 \* x4;  double x16 = x8 \* x8;  double x17 = x16 \* x;  double x5 = x4 \* x;  \*pX2 = x2;  \*pX5 = x5;  \*pX17 = x17;  return S\_OK;  }  public:  CComServerTest() : \_cRef(1) {  InterlockedIncrement(&g\_cRefDll); // g\_cRefDll++;  }  private:  LONG \_cRef;  ~CComServerTest(void) {  InterlockedDecrement(&g\_cRefDll); // g\_cRefDll--;  }  };  //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  // Creation function  HRESULT CComServerTest\_CreateInstance(\_\_in REFIID riid, \_\_out void \*\*ppv) {  HRESULT hr = E\_OUTOFMEMORY;  CComServerTest \*pInstance = new CComServerTest();  if (pInstance != nullptr) {  hr = pInstance->QueryInterface(riid, ppv);  pInstance->Release();  }  return hr;  }  //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

**CComServiceTest.cpp**

|  |
| --- |
| //  // A class that implements IComTest.  //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  #include "pch.h"  #pragma hdrstop  #include <objbase.h>  #include <shlwapi.h>  #include <assert.h>  #include <IComTest\_h.h>  #include "ComSampleServiceGuids.h"  #include "ComSampleServiceCreateInstances.h"  class CComServiceTest : public IComTest, public ILPV  {  public:  // IUnknown    IFACEMETHODIMP\_(ULONG) AddRef()  {  return InterlockedIncrement(&\_cRef);  }    IFACEMETHODIMP\_(ULONG) Release()  {  assert(\_cRef > 0);    ULONG cRef = InterlockedDecrement(&\_cRef);  if (0 == cRef)  {  delete this;  }  return cRef;  }  IFACEMETHODIMP QueryInterface(\_\_in REFIID riid, \_\_out void \*\*ppv)  {  static const QITAB qit[] =  {  QITABENT(CComServiceTest, IComTest),  QITABENT(CComServiceTest, ILPV),  { 0 },  };  return QISearch(this, qit, riid, ppv);  #if 0  // ANOTHER POSSIBLE IMPLEMENTATION FOR QueryInterface METHOD.  HRESULT hr = (ppv != nullptr) ? S\_OK : E\_INVALIDARG;  if (SUCCEEDED(hr))  {  \*ppv = nullptr;  hr = E\_NOINTERFACE;  if (\_\_uuidof(IComTest) == riid)  {  \*ppv = static\_cast<IComTest\*>(this);  hr = S\_OK;  }  else if (\_\_uuidof(IUnknown) == riid)  {  \*ppv = static\_cast<IUnknown\*>(this);  hr = S\_OK;  }  if (SUCCEEDED(hr))  {  reinterpret\_cast<IUnknown\*>(\*ppv)->AddRef();  }  }  return hr;  #endif  }  // IComTest  IFACEMETHODIMP WhoAmI(\_Out\_ LPWSTR \*ppwszWhoAmI)  {  HRESULT hr = (ppwszWhoAmI != nullptr) ? S\_OK : E\_INVALIDARG;  if (SUCCEEDED(hr))  {  // Create a message string the contains the name of the current process.  wchar\_t wszProcessName[MAX\_PATH] = {};  DWORD dwResult = GetModuleFileNameW(nullptr, wszProcessName, ARRAYSIZE(wszProcessName));  hr = ((dwResult < ARRAYSIZE(wszProcessName)) && (dwResult != 0)) ? S\_OK : HRESULT\_FROM\_WIN32(GetLastError());  if (SUCCEEDED(hr))  {  LPCWSTR pwszMessagePreface = L"Com Test Service running in process ";    // Prepare the message.  size\_t cchMessagePlusTerminatingNul = wcslen(pwszMessagePreface) + wcslen(wszProcessName) + 1 /\* For the terminating NUL character \*/;  wchar\_t \*pwszMessage = (wchar\_t \*)CoTaskMemAlloc(cchMessagePlusTerminatingNul \* sizeof(wchar\_t));  hr = (pwszMessage != nullptr) ? S\_OK : E\_OUTOFMEMORY;  if (SUCCEEDED(hr))  {  hr = StringCchPrintfW(pwszMessage, cchMessagePlusTerminatingNul, L"%s%s", pwszMessagePreface, wszProcessName);  if (SUCCEEDED(hr))  {  hr = SHStrDupW(pwszMessage, ppwszWhoAmI);  }  CoTaskMemFree(pwszMessage);  }  }  }    return hr;  }  IFACEMETHODIMP CalculatePowers(  /\* [in] \*/ double x,  /\* [out] \*/ double\* pX2,  /\* [out] \*/ double\* pX5,  /\* [out] \*/ double\* pX17) override  {  if (pX2 == nullptr || pX5 == nullptr || pX17 == nullptr)  {  return E\_POINTER;  }  double x2 = x \* x;  double x4 = x2 \* x2;  double x8 = x4 \* x4;  double x16 = x8 \* x8;  double x17 = x16 \* x;  double x5 = x4 \* x;  \*pX2 = x2;  \*pX5 = x5;  \*pX17 = x17;  return S\_OK;  }  public:  CComServiceTest() : \_cRef(1)  {  }  private:  LONG \_cRef;    ~CComServiceTest(void)  {  }  };  //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  // Creation function  HRESULT CComServiceTest\_CreateInstance(\_\_in REFIID riid, \_\_out void \*\*ppv)  {  HRESULT hr = E\_OUTOFMEMORY;  CComServiceTest \*pInstance = new CComServiceTest();  if (pInstance != nullptr)  {  hr = pInstance->QueryInterface(riid, ppv);  pInstance->Release();  }  return hr;  }  //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

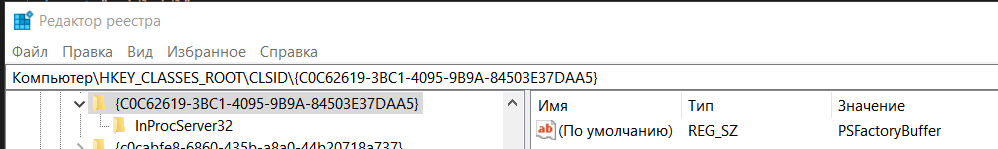
**Main.cpp**

|  |
| --- |
| #include <windows.h>  #include <combaseapi.h>  #include <strsafe.h>  #include <initguid.h>  #include "IComTest\_h.h"  #include <ComSampleServerGuids.h>  #include <ComSampleServiceGuids.h>  #include <winerror.h>  HRESULT Execute(\_In\_ const IID &rclsid, \_In\_ DWORD dwCoInit, \_In\_ DWORD dwClsContext)  {  HRESULT hr = CoInitializeEx(NULL, dwCoInit);  if (SUCCEEDED(hr))  {  IComTest \*pComTest;  hr = CoCreateInstance(rclsid,  NULL,  dwClsContext,  IID\_PPV\_ARGS(&pComTest));  if (SUCCEEDED(hr))  {  LPWSTR pwszWhoAmI = nullptr; // Инициализируем  HRESULT hrWhoAmI = pComTest->WhoAmI(&pwszWhoAmI);  if (SUCCEEDED(hrWhoAmI))  {  wprintf(L"%s. Client calling from %s. COM Server running %s.\n",  pwszWhoAmI ? pwszWhoAmI : L"<NULL>",  (dwCoInit == COINIT\_MULTITHREADED) ? L"MTA" : L"STA",  (dwClsContext == CLSCTX\_INPROC\_SERVER) ? L"in-process" : L"out-of-process");  CoTaskMemFree(pwszWhoAmI);  }  else  {  wprintf(L"Error: WhoAmI failed with hr=0x%08X\n", hrWhoAmI);  }  ILPV\* pLPV = nullptr;  HRESULT hrQI = pComTest->QueryInterface(IID\_ILPV, (void\*\*)&pLPV);  if (SUCCEEDED(hrQI))  {  double x = 2.0;  double x2 = 0.0, x5 = 0.0, x17 = 0.0;  HRESULT hrCalc = pLPV->CalculatePowers(x, &x2, &x5, &x17);  if (SUCCEEDED(hrCalc))  {  wprintf(L" Calculations for x=%.2f: x^2=%.6f, x^5=%.6f, x^17=%.6f\n",  x, x2, x5, x17);  }  else  {  wprintf(L" Error: CalculatePowers failed with hr=0x%08X\n", hrCalc);  }  pLPV->Release();  pLPV = nullptr;  }  else  {  wprintf(L" Error: QueryInterface for ILPV failed with hr=0x%08X. Component might not implement it.\n", hrQI);  }  pComTest->Release();  pComTest = nullptr;  }  CoUninitialize();  }  return hr;  }  int \_\_cdecl main()  {  Execute(CLSID\_CComServerTest, COINIT\_MULTITHREADED, CLSCTX\_INPROC\_SERVER);  Execute(CLSID\_CComServerTest, COINIT\_MULTITHREADED, CLSCTX\_LOCAL\_SERVER);  Execute(CLSID\_CComServerTest, COINIT\_APARTMENTTHREADED, CLSCTX\_INPROC\_SERVER);  Execute(CLSID\_CComServerTest, COINIT\_APARTMENTTHREADED, CLSCTX\_LOCAL\_SERVER);  Execute(CLSID\_CComServiceTest, COINIT\_MULTITHREADED, CLSCTX\_LOCAL\_SERVER);  Execute(CLSID\_CComServiceTest, COINIT\_APARTMENTTHREADED, CLSCTX\_LOCAL\_SERVER);  return 0;  } |

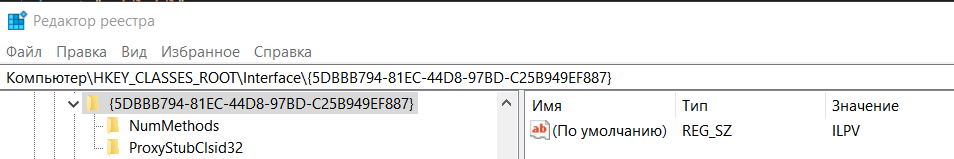


Интерфейсы

**IComTest**

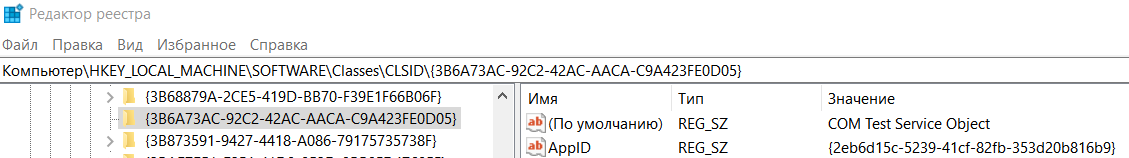


**ILPV**



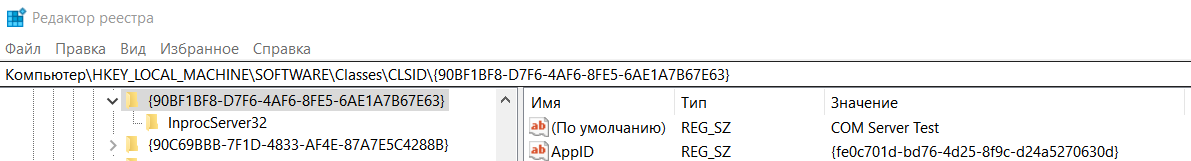
**ComSampleServiceGuids.h**



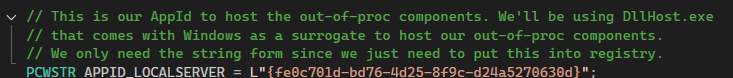


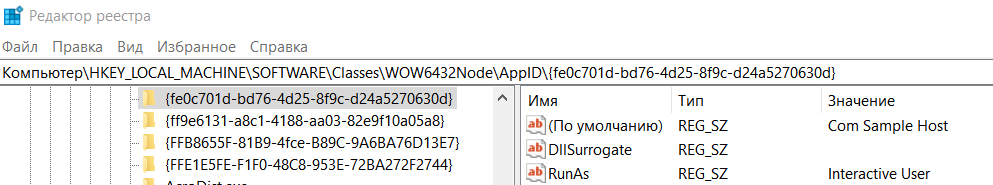
**ComSampleServerGuids.h**



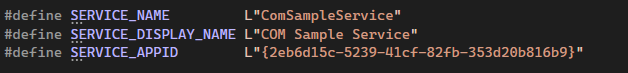


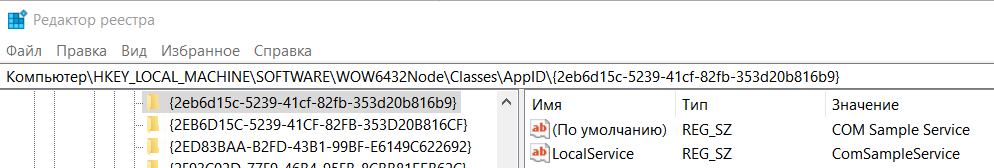
**Dll.cpp**



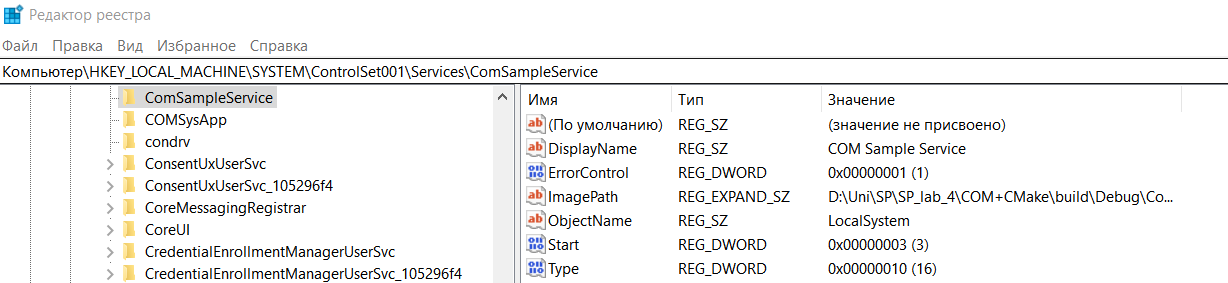


**Infrastructure.h**





**ComSampleService**



**Вопросы для контроля:**

1. **Что такое Component Object Model (далее - COM)?**

**Component Object Model** (COM) — объектная модель компонентов от Microsoft, предназначенная для проектирования и создания модулей, независимых от языка программирования. COM реализована во всех версиях Windows, а в других системах (например, UNIX или macOS) может поддерживаться сторонними средствами. Модель развивалась как продолжение подходов к модульной разработке на C++.

Одна из ключевых особенностей COM — это предоставление общего двоичного стандарта для компонентов. Благодаря ему модули, созданные на разных языках программирования и под разные операционные системы, могут взаимодействовать без перекомпиляции или изменения кода. Это особенно важно для повторного использования программного обеспечения в разных средах.

Компоненты можно распространять в виде двоичных файлов (например, DLL или EXE), и они будут работать независимо от языка или системы разработки.

Другое важное свойство СОМ известно под названием независимости от местоположения. Независимость от местоположения означает, что пользователь компонента, клиент, не обязательно должен знать, где находится определенный компонент

Компонент может находиться непосредственно в адресном пространстве задачи клиента (DLL-файл), в пространстве другой задачи на том же компьютере (ЕХЕ-файл) или на компьютере, расположенном за сотни миль (распределенный объект).

Чтобы понять COM (и, следовательно, все технологии, основанные на COM), важно понимать, что это не объектно-ориентированный язык, а стандарт

Ключевая идея – разделение интерфейса и реализации. Клиент работает только с интерфейсами, не зная деталей реализации объекта.

Единственным языковым требованием COM является то, что код генерируется на языке, который может создавать структуры указателей и, явно или неявно, вызывать функции с помощью указателей

1. **Что такое COM-компонент?**

Соответственно, первым основным понятием, которым оперирует стандарт СОМ, является COM-компонент (COM-объект), представляющий собой программный модуль.

COM-объект можно сравнить с объектом в понимании С++ или Java. Объект СОМ – это некоторая сущность, имеющая состояние и методы доступа, позволяющие изменять это состояние.

СОМ-объекты можно создавать прямым вызовом специальных функций, но напрямую уничтожить его невозможно. Вместо прямого уничтожения используется механизм самоуничтожения, основанный на подсчете ссылок.

1. **Что такое COM-интерфейс?**

COM-компонент (или COM-класс): Это реализация одного или нескольких **COM-интерфейсов**. По сути, это код, выполняющий определенную логику и предоставляющий доступ к ней через стандартные интерфейсы (IUnknown и другие).

1. **Что такое COM-контейнер? Какие бывают?**

COM-контейнер – это место, где размещаются COM-компоненты в Windows.

Существует два вида контейнеров:

* DLL-файл (для in-process серверов)
* EXE-файл (для local и remote out-of-process серверов)

1. **Что такое COM-сервер и COM-клиент? Какие бывают сервера?**

**COM-клиент** – это приложение, которое использует COM-компоненты (вызывает функции их интерфейсов).

**COM-сервер**– это контейнер (DLL или EXE) с расположенными в нем COM-компонентами, который предоставляет реализацию этих компонентов.

Типы серверов:

По типу контейнера и расположению:

* **INPROC**(In-process): Компоненты находятся в DLL и загружаются в адресное пространство клиента (локальный).
* **LOCAL**(Out-of-process): Компоненты находятся в EXE-файле, работающем как отдельный процесс на том же компьютере (локальный).
* **REMOTE**(Out-of-process): Компоненты находятся в EXE-файле, работающем на удаленном компьютере (удаленный).

По количеству компонентов:

* **Однокомпонентные**: Сервер реализует только один COM-компонент.
* **Многокомпонентные**: Сервер реализует два или более COM-компонентов

Есть механизм запуска in-process сервера (DLL) в суррогатном процессе EXE для получения преимуществ out-of-process выполнения.

1. **Что такое GUID? Для чего он используется?**

**GUID**(Globally Unique Identifier) – это глобально уникальный 128-битный идентификатор. Его уникальность достигается использованием информации о компьютере (например, MAC-адрес) и времени создания.

Используется в COM для однозначной идентификации:

* Классов компонентов (CoClass), где он называется CLSID (Class Identifier).
* Интерфейсов, где он называется IID (Interface Identifier).
* Библиотек типов (Type Libraries).  
   Это позволяет клиентам и системе точно находить и запрашивать нужные классы и интерфейсы.

1. **Какие бывают интерфейсы?**

Согласно лекции, интерфейсы бывают двух типов:

**Стандартные**: Имеют предопределенные GUID-идентификаторы. Важнейшим является IUnknown.

**Произвольные** (Custom): Определяются разработчиками. Все произвольные интерфейсы должны прямо или косвенно наследоваться от IUnknown.

1. **Поясните функции интерфейса IUnknown.**

**IUnknown**– это базовый интерфейс, от которого должны наследоваться все COM-интерфейсы. Он предоставляет три основные функции (метода) для управления жизненным циклом объекта и навигации по интерфейсам:

**QueryInterface**(REFIID iid, void \*\*ppvObject): Позволяет запросить у объекта указатель на другой интерфейс, поддерживаемый этим объектом, по его IID. Если интерфейс поддерживается, метод увеличивает счетчик ссылок и возвращает указатель.

**AddRef**(): Увеличивает внутренний счетчик ссылок на интерфейс на 1. Вызывается каждый раз, когда клиент получает новый указатель на интерфейс.

**Release**(): Уменьшает счетчик ссылок на интерфейс на 1. Вызывается, когда клиент прекращает использование указателя на интерфейс. Когда счетчик достигает нуля, объект может быть уничтожен.

1. **Объясните «счётчик ссылок на интерфейс». Когда увеличивается/уменьшается?**

«Счётчик ссылок на интерфейс» – это внутренний счетчик, который ведет каждый COM-объект для отслеживания того, сколько клиентов в данный момент используют указатели на его интерфейсы. Это основной механизм управления временем жизни объекта в COM.

Счетчик увеличивается (на 1) при вызове метода AddRef(). Это происходит, когда клиент успешно получает указатель на интерфейс (например, через CoCreateInstance или QueryInterface).

Счетчик уменьшается (на 1) при вызове метода Release(). Это происходит, когда клиент больше не нуждается в указателе на интерфейс и освобождает его.

Когда счетчик достигает нуля, объект понимает, что на него больше нет ссылок, и может безопасно самоуничтожиться.



1. **Поясните функции интерфейса IClassFactory.**

**IClassFactory**– это стандартный интерфейс, реализуемый фабрикой классов. Фабрика классов отвечает за создание экземпляров конкретного COM-компонента (CoClass). Основные функции (методы) интерфейса (Слайд 30):

**CreateInstance**(IUnknown \*pUnkOuter, REFIID riid, void \*\*ppvObject): Создает новый экземпляр COM-компонента, ассоциированного с этой фабрикой, и возвращает указатель на запрошенный интерфейс (riid) этого нового экземпляра. Параметр pUnkOuter используется для агрегации.

**LockServer**(BOOL fLock): Увеличивает (fLock = TRUE) или уменьшает (fLock = FALSE) счетчик блокировок сервера. Это позволяет клиенту предотвратить выгрузку сервера (особенно DLL) из памяти, даже если в данный момент нет активных экземпляров компонентов.

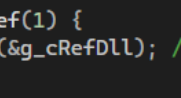
1. **Объясните «счётчик ссылок на компонент». Когда увеличивается/уменьшается?**

Этот счетчик отслеживает общее количество активных экземпляров всех компонентов, предоставляемых данным сервером.

Он увеличивается, когда создается новый экземпляр компонента (обычно в конструкторе компонента, который вызывается через IClassFactory::CreateInstance).

Он уменьшается, когда экземпляр компонента уничтожается (обычно в деструкторе компонента).

Этот счетчик используется сервером (особенно DLL) для определения, можно ли его выгрузить из памяти (например, в функции DllCanUnloadNow). Он отличен от «счетчика ссылок на интерфейс», который ведется каждым объектом индивидуально.



1. **Какие соглашения о вызове и возврате должны соблюдаться для COM-функций?**

**Соглашение о вызове:** Все методы COM-интерфейсов должны использовать соглашение stdcall.

**Тип возвращаемого значения:** Почти все методы должны возвращать значение типа HRESULT для индикации успеха или кода ошибки. Исключениями являются методы IUnknown::AddRef и IUnknown::Release, которые возвращают ULONG (текущее значение счетчика ссылок).

1. **Опишите структуру HRESULT.**

**HRESULT**– это 32-битное целое число, используемое для возврата кодов состояния в COM. Его структура:

**Бит 31** (Severity): 0 для успеха (SUCCEEDED), 1 для ошибки (FAILED). (Слайд 50 уточняет: биты 31-30 кодируют степень серьезности: 00=Success, 01=Informational, 10=Warning, 11=Error).

**Бит 29** (Customer): 0 для кодов Microsoft/системных, 1 для пользовательских кодов.

**Бит 28** (Reserved): Зарезервирован.

**Биты 16-27** (Facility): Код, идентифицирующий источник ошибки (например, FACILITY\_WIN32, FACILITY\_STORAGE, FACILITY\_RPC, FACILITY\_ITF и т.д.).

**Биты 0-15** (Code): Конкретный код состояния или ошибки в рамках указанного Facility.

Для проверки HRESULT используются макросы SUCCEEDED() и FAILED().

1. **Что должен знать COM-клиент чтобы взаимодействовать с COM-сервером?**

CLSID компонента, экземпляр которого он хочет создать.

IID интерфейсов, которые он собирается использовать.

Тип сервера (in-proc, local, remote - это влияет на контекст создания).

Структуры (сигнатуры) методов используемых интерфейсов (т.е., какие параметры они принимают и что возвращают).

1. **Что такое regsvr32? Принцип работы?**

**regsvr32.exe**– это стандартная утилита Windows для регистрации и отмены регистрации COM-серверов (в основном, in-process DLL) в системном реестре.

**Принцип работы:**

При вызове regsvr32 <имя\_dll>: утилита загружает указанную DLL в память и вызывает ее экспортируемую функцию DllRegisterServer(). Эта функция должна создать в реестре все необходимые записи (CLSID, ProgID, путь к серверу, модель потоков и т.д.).

При вызове regsvr32 /u <имя\_dll>: утилита загружает DLL и вызывает ее экспортируемую функцию DllUnregisterServer(). Эта функция должна удалить из реестра все записи, созданные DllRegisterServer().

Регистрация необходима для реализации свойства «независимости от местоположения», так как она позволяет системной библиотеке COM (OLE32.dll) находить и загружать сервер по его CLSID.

1. **Где найти информацию о компоненте в реестре?**

Информация о зарегистрированных COM-компонентах хранится в системном реестре Windows, в основном, в разделе HKEY\_CLASSES\_ROOT\CLSID.

Каждый компонент имеет свой подраздел, имя которого соответствует его CLSID в формате {XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}.

Внутри этого подраздела находятся другие подразделы, описывающие компонент, например:

InprocServer32: Для DLL-серверов, содержит путь к файлу DLL и модель потоков (ThreadingModel).

LocalServer32: Для EXE-серверов, содержит путь к файлу EXE.

ProgID: Человекочитаемый идентификатор.

1. **Перечислите и объясните 5 COM-функций экспортируемых из DLL.**

COM DLL-сервер должен экспортировать несколько стандартных функций для взаимодействия с COM-инфраструктурой:

**DllGetClassObject**(REFCLSID rclsid, REFIID riid, LPVOID \*ppv): Вызывается OLE32.DLL для получения объекта фабрики классов для указанного rclsid. Возвращает указатель на запрошенный интерфейс фабрики (riid, обычно IID\_IClassFactory). Это основная точка входа для создания компонентов из DLL.

**DllCanUnloadNow**(): Вызывается OLE32.DLL, чтобы спросить у DLL, можно ли ее безопасно выгрузить из памяти. DLL должна вернуть S\_OK (можно), только если нет активных объектов и блокировок сервера (счетчики экземпляров и блокировок равны нулю), иначе S\_FALSE (нельзя).

**DllRegisterServer**(): Вызывается утилитой regsvr32 (без /u). Функция должна записать в реестр всю информацию о компонентах, реализованных в DLL.

**DllUnregisterServer**(): Вызывается утилитой regsvr32 /u. Функция должна удалить из реестра всю информацию о компонентах DLL.

**DllInstall**(BOOL bInstall, PCWSTR pszCmdLine): (Менее обязательная, но часто присутствует) Вызывается regsvr32 /i для выполнения дополнительных действий по установке/удалению, не связанных напрямую со стандартной COM-регистрацией.

Замечание: DllMain тоже экспортируется, но это стандартная точка входа для любой Windows DLL, а не специфичная функция COM-инфраструктуры.

1. **Какая системная библиотека отвечает за COM-инфраструктуру?**

Основной системной библиотекой, отвечающей за COM-инфраструктуру в Windows, является OLE32.DLL. Она предоставляет API функции COM (например, CoInitializeEx, CoCreateInstance, CoUninitialize), загружает серверы, управляет подсчетом ссылок (частично), маршалингом (при необходимости) и другими аспектами работы COM. При компиляции используется библиотека импорта OLE32.LIB .