# Создание библиотеки класса в Visual C#(Sharp)

Наш третий пример не является приложением .NET; это разделяемая функциальность, которая обычно называется *библиотекой класса* (class library). Приления Windows и консольные приложения можно выполнить  в  Проводнике Windows или из командной строки. Но библиотеку класса запустить на исполнение пользователь не может; это можно только сделать из приложения этих двух типов. Библиотека класса является удобным хранилищем для кода, который используется в нескольких приложениях.

#### Добавление библиотеки класса в решение

Приступим к созданию библиотеки класса, которая может совместно использаться в нашем приложении Windows и консольном приложении. Чтобы добавить новый проект, составляющий библиотеку класса, в решение ThreeExamples, волните такую последовательность действий:

1.  Щелкните правой кнопкой  мыши  по имени  решения  ThreeExamples в **Solution Explorer.**

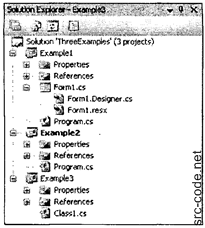
**2.  В** открывшемся контекстном меню выберите пункты **Add | New Project.**

3.  В панели **Templates** открывшегося окна **Add New Project** выберите компонент

**Class Library.** В поле **Name** измените имя приложения на Exampie3.

Добавленный проект должен отобразиться в решении (рис. 1.10).

2  В  русских   версиях   Windows   сообщение   будет   "Для   продолжения   нажмите   любую   клаву. ..". — *Пер.*



**Рис. 1.10.** Структура решения, содержащего все три проекта

Проект   Exampie3   содержит  единственный  файл  Classl.cs,   который  является   пртым  файлом  исходного кода.

#### Перемещение функциональности

Теперь мы переместим код, ответственный за вывод  текста  "hello,  world",  из Exampie2 в Exampie3. Для этого вставьте в исходный код в файле Classl.cs код, веленный  жирным  шрифтом:

using System;

using System.Collections.Generic; using System.Text;

namespace Example3

{

public class Classl

{

**public static void HelloWorldO**

**{**

**Console.WriteLine("hello, world");**

}

}

}

Вставленный код содержит метод HeiioWorid\*). При вызове этого метода  он  водит текст "hello, world". Как было сказано ранее в этой главе, метод представляет собой набор инструкций для выполнения  определенной  задачи.  Более  подробно методы  рассматриваются  в *главе  2.*

Для того чтобы приложения могли совместно использовать код библиотеки класса, необходимо, чтобы проекты знали о существовании друг друга. Это достигается посредством ссылок.

#### Определение ссылок

Чтобы один проект знал об определениях в другом проекте, необходимо опредить *ссылку.* Концепция ссылки заключается в том, чтобы указать, что проект знает о другой функциональности.

***ПРИМЕЧАНИЕ***

Проект знает только о функциональности, которая было объявлена открытой (public). Открытая функциональность, или как еще говорят программисты  в  С#  *открытая  оасть видимости,* получается в результате объявления  типа  с  помощью  ключевого слова public. Открытая область видимости и другие типы областей видимости рассматриваются на протяжении всей книги.

Чтобы проект Exampie2 знал о функциональности, содержащейся в файле Classl.cs, необходимо установить физическую ссылку следующим образом:

**1.** Разверните узел **References** проекта Exampie2, щелкнув по его значку со знаком "плюс". Обратите внимание, что уже существуют три ссылки. Когда вы ввели текст console.writeLine() в код файла Classl.cs, то использовали функциальность, предоставляемую пространством имен system.

2. Щелкните  правой  кнопкой  мыши  по  элементу  **References**и  выберите  опцию

**Add Reference.**

3. Щелкните по вкладке **Projects.**

4. Выберите Exampie3, после чего нажмите кнопку **ОК.** В результате этих дейсий проект Example3 будет добавлен В ССЫЛКИ проекта Example2.

После установки ссылки проект Example2 может вызывать функциональность пркта Example3.

***ПРИМЕЧАНИЕ***

В файле Class1.cs первые три строчки начинаются с ключевого слова using. Оно сбщает Visual С# Express, что вы хотите использовать функциональность, определеую в ссылке на ресурс после ключевого слова using. В этом примере мы не исполовали этот быстрый способ создания ссылки на функциональность, с тем, чтобы показать другой способ ее создания.

Источник: Гросс  К. С# 2008:  Пер. с англ. — СПб.:  БХВ-Петербург, 2009. — 576 е.:  ил. — (Самоучитель)

[Tweet](http://twitter.com/share)

[Нравится](http://connect.mail.ru/share)

Tags: [класса](http://src-code.net/tag/klassa/), [которая](http://src-code.net/tag/kotoraya/), [приложения](http://src-code.net/tag/prilozheniya/), [проект](http://src-code.net/tag/proekt/), [ссылки](http://src-code.net/tag/ssylki/), [функциональности](http://src-code.net/tag/funkcionalnosti/), [функциональность](http://src-code.net/tag/funkcionalnost/)

#### По теме:

### [Генерация кода XML - Введение](http://src-code.net/generaciya-koda-xml-vvedenie/)

### [Включение условий в выражения if](http://src-code.net/vklyuchenie-uslovij-v-vyrazheniya-if/)

### [Создание документов связанных гиперссылками](http://src-code.net/sozdanie-dokumentov-svyazannyx-giperssylkami/)

### [Выбор базового класса](http://src-code.net/vybor-bazovogo-klassa/)

### [Отладка компонентов в режиме разработки](http://src-code.net/otladka-komponentov-v-rezhime-razrabotki/)

### [Создаем проект](http://src-code.net/sozdaem-proekt/)

### [Контекстное меню компонента](http://src-code.net/kontekstnoe-menyu-komponenta/)

# ак создать библиотеку классов в Microsoft Visual Studio 2013

[admin](http://mvblog.ru/author/admin/) • [17.10.2014](http://mvblog.ru/archives/817/) • [Оставить комментарий](http://mvblog.ru/archives/817/#respond)

Вам никогда не приходилось использовать ранее написанный класс в последующих проектах? Мне — приходилось, да и не один раз. Однако каждый раз добавлять один и тот же класс в проект очень утомительно (я сейчас имею в виду обычную вставку файла, а порой и нескольких файлов, с нужным классом). Мало того, что это утомительно, так еще и неудобно. Неудобства возникают при внесении изменений в такие классы. Ведь добавив что-либо в класс, изменения будут применены **только к этому классу.** Иными словами, мы получим два различных класса, назначения которых будут одинаковыми:

* исходный класс, который был взят для нового проекта;
* новый класс, который является копией исходного, но с какими-либо изменениями.

Чтобы избежать подобного, необходима, так сказать, «ссылка на класс», которая будет использоваться во всех проектах, и если нужно будет внести какие-либо изменения, то они будут применены ко всем проектам, использующие эту самую «ссылку». Такой принцип работы мы можем получить, создав библиотеку классов и подключив ее ко всем нужным проектам.

## Итак, начнем...

Из названия понятно, что создавать библиотеку я буду в Microsoft Visual Studio 2013. Как только VS запустилась, необходимо создать проект. Нажимаем Файл > Создать > Проект… (или просто Ctrl + Shift + N). Затем выбираем язык программирования, на котором будет написана библиотека (я буду писать на C#). После чего из списка выбираем «Библиотека классов», даем имя библиотеке и выбираем ее расположение.

## Написание кода

Как только проект создан, приступаем к написанию самого кода. Я напишу простой класс **Test**, в котором будет одно всего одно поле и один простой метод.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TestLibrary

{

public class Test

{

string message;

public Test(string message)

{

this.message = message;

}

// Выводит в консоль строку message

public void ShowMessage()

{

Console.WriteLine(message);

}

}

}

Теперь остается только скомпилировать библиотеку. Для этого нажимаем Сборка > Собрать решение… (или просто клавишу *F6*).

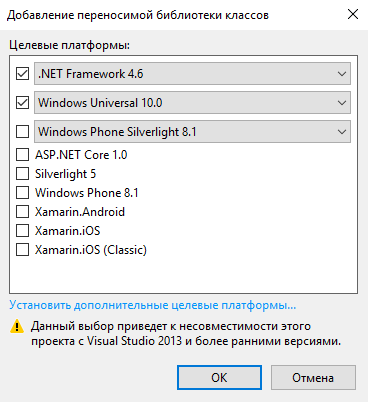
Вот и все! Мы создали библиотеку, которую можно использовать в нескольких проектах. При этом, если понадобится изменить что-то, например метод **ShowMessage**, то теперь не придется изменять этот метод во всех проектах, а лишь изменить его в библиотеке и вновь ее скомпилировать.

Повторное использование кода и совместимость под различные платформы в наше время довольно актуальный вопрос. Кроме того MSFT в последнее время старается угодить разработчикам различных платформ и языков программирования. Именно поэтому, вероятно, возможности универсальной платформы Windows позволяют использовать не один тип библиотек. Под катом о том, какие библиотеки, как и зачем можно использовать в приложениях UWP.

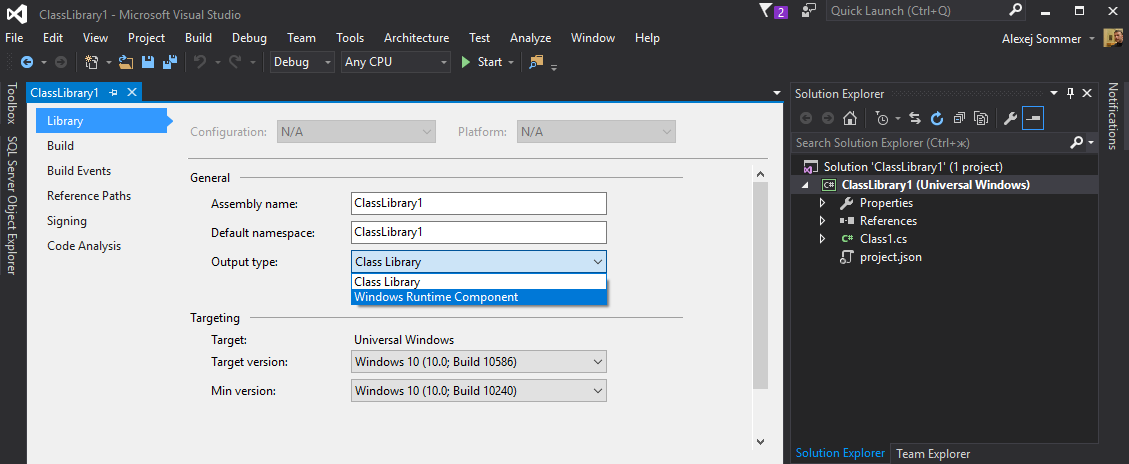
**Библиотека классов — Class Library для приложений Windows Store**

.Net библиотека с классами и методами, доступными для приложений Windows Store apps. Может быть использована только UWP приложениями, написанными на языках .Net. То есть, JavaScript, например, отпадает.   
  
Заметьте, что обычные .Net библиотеки классов UWP приложение само собой использовать не может.  
Список отличий и изменений, которые необходимо сделать при конвертации кода .Net в .Net для приложений Windows Store вы можете найти по ссылке: [.NET for Windows Store apps overview](https://msdn.microsoft.com/en-us/library/br230302.aspx)  
  
Изменений и различий довольно много, хотя наблюдается и определенное сходство. Зачастую отличаются только пространства имен, а методы и свойства остаются прежними. Это сделано для удобства портирования существующего кода. Разумеется, не весь функционал .Net доступен в приложениях Windows Store. Отчасти из соображений безопасности или же из-за того, что его концептуально не захотели дублировать в UWP.

**Библиотека классов переносимая — PCL (Portable Class Library)**

В этих библиотеках доступна определенная часть возможностей в зависимости от того, какие платформы вы выбрали. Целевые платформы выбираются при создании проекта  
  
  
  
Эти целевые платформы можно изменить в любой момент в свойствах проекта.  
Так как код будет один для нескольких платформ, то все типы и члены должны присутствовать в выбранных платформах и не должны иметь никакой специфики платформы. Также они не должны быть deprecated или помеченными на исключение.

**Компонент среды выполнения Windows -Windows Runtime Component**

Компонент, который может использоваться с любым из языков платформы WinRT. Расширение у компонента оканчивается на .winmd. Но пускай вас это не путает, компонент очень часто все-таки используется как библиотека, а метаданные, которые он содержит позволяют ему работать со всеми языками WinRT. Это и является основным плюсом.  
  
Для примера могу показать вам как в созданном проекте Class Library через свойства можно выбрать Runtime Component в качестве результата построения:  
  
  
  
Но в этом случае вы потеряете кросс-языковую совместимость и сможете работать с такой библиотекой только из C# и VB.  
  
Runtime Component основан на COM. Выходит, что имеется возможность написать нативный Windows Runtime компонент на managed языке.  
Во всех языках WinRT используются свои типы данных. Но в Runtime компоненте данные должны быть унифицированы.  
Примитивные типы более-менее сходны во всех языках. Отличается большей частью только JavaScript.   
Таблицу сопоставления можно найти по ссылкам:  
[Windows Runtime base data types](https://msdn.microsoft.com/ru-ru/library/windows/desktop/br205768(v=vs.85).aspx)  
[.NET Framework mappings of Windows Runtime types](https://msdn.microsoft.com/en-us/windows/uwp/winrt-components/net-framework-mappings-of-windows-runtime-types)  
  
Языки WinRT различаются и поэтому иногда необходимо приходить к какому-то общему знаменателю. Например, в .Net и JavaScript приложениях строки неизменяемые (immutable), а в C++ они mutable. В компоненте Windows Runtime строки immutable.  
  
Ограничения/правила:

* Публичные элементы должны быть обязательно запечатаны (sealed).
* Публичные классы или интерфейсы не могут быть generic или реализовывать не WinRT интерфейс.
* Публичные классы не могут быть производными от WinRT типов.
* Все поля, параметры и возвращаемые значения публичных членов должны быть типа Windows Runtime.
* Коренное пространство имен должно соответствовать названию сборки и не может начинаться с «Windows».
* WRC не поддерживает полиморфизм
* Методы в Windows Runtime могут быть перегружены, но в случае одинакового количества параметров они должны быть помечены атрибутом. Официальный пример:

public string OverloadExample(string s)

{

return s;

}

[Windows.Foundation.Metadata.DefaultOverload()]

public int OverloadExample(int x)

{

return x;

}

В Class Library для приложений Windows Store меньше всего ограничений, поэтому если у вас приложение на .Net языках, то лучше выбирать эту библиотеку.

**Практический пример:**

Если ваше UWP приложение использует WebView для отображения веб-сайтов, то у вас есть возможность использовать в приложении класс, который будет доступен из JavaScript-а загруженных веб страниц.  
И вот этот класс должен находится внутри компонента Windows Runtime. Вот где необходима кросс-языковая совместимость.  
  
Создается компонент, и из основного проекта, содержащего страницу WebView на него делается ссылка. Внутри компонента публичный класс или классы могут быть помечены атрибутом [AllowForWeb]. Все публичные методы таких классов будут доступны из JavaScript загруженной в WebView страницы. Необходимо только при начале навигации на страницу добавлять этот класс с помощью метода [AddWebAllowedObject](https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.controls.webview.addweballowedobject.aspx).  
  
Да, конечно, есть вариант использовать window.external.notify в коде JS, но имеются определенные ограничения безопасности, которые разрешают выполнение этих вызовов только с доверенных https сайтов.   
  
Немного повторюсь и продублирую пример с MSDN. В класс WinRT компонента добавим метод SomeValueFromJS

using Windows.Foundation.Metadata;

namespace MyRuntimeComponent

{

[AllowForWeb]

public sealed class MyNativeClass

{

public void SomeValueFromJS(string value)

{

}

}

}

В XAML добавим WebView

<WebView x:Name="webView" Source="https:// www.contoso.com/script.htm"

NavigationStarting="webView\_NavigationStarting"/>

И при начале навигации внедрим наш класс в JavaScript

private void webView\_NavigationStarting(WebView sender, WebViewNavigationStartingEventArgs args)

{

if (args.Uri.Host == "www.contoso.com")

{

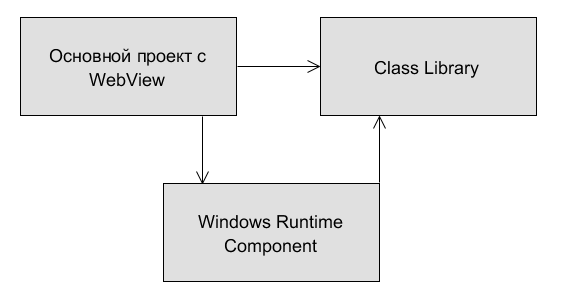
webView.AddWebAllowedObject("csharpclass", new MyNativeClass());

}

}

Теперь из кода JS можно вызвать этот метод так:

csharpclass.nativeMethod("hello!");

Казалось бы все нормально, но вот незадача, зачастую может возникать необходимость передавать данные на страницу с WebView. Проект с WebView уже ссылается на WinRT компонент и поэтому обратную ссылку нам не добавить из-за циклической зависимости. Ограничения WRC не позволяют нам использовать некоторые варианты. В качестве решения можно создать промежуточный слой в виде библиотеки Class Library для приложений Windows Store у которой ограничений гораздо меньше. Ссылку на эту библиотеку содержащую класс с названием BridgeClass можно сделать и из основного проекта и с проекта компонента Windows Runtime.   
  
  
  
Возможности .Net позволят нам создать доступный ивент

public static event Action<string>

и подписаться на событие в основном проекте. Давайте рассмотрим в виде кода. Код класса BridgeClass:

public class BridgeClass

{

public static event Action<string> MessageReceived;

public static void Broadcast(string message)

{

if (MessageReceived != null) MessageReceived(message);

}

}

В основном проекте, в code-behind страницы с WebView подписываемся на событие:

BridgeClass.MessageReceived += ShowMessage;

И реализуем:

void ShowMessage(string msg)

{

}

Теперь из метода, доступного для JS можем передать значение в C#:

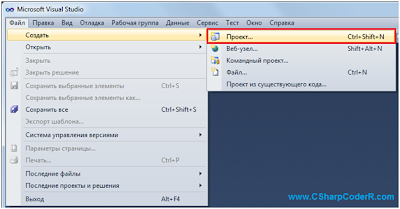
public void SomeValueFromJS(string value)

{

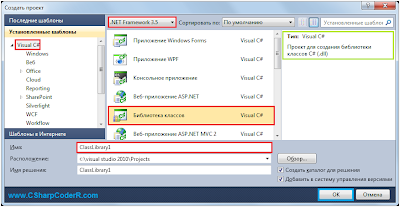
BridgeClass.Broadcast(value);

}

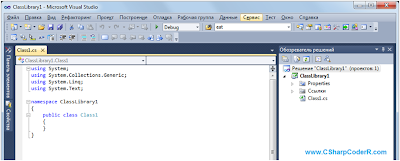
Иногда при запуске какой-либо программы появляется сообщение, что не найден файл **\*.dll**. Для операционных систем Microsoft Windows, большая часть функциональных возможностей операционной системы обеспечивается библиотеками динамической компоновки (DLL). Кроме того, некоторые возможности программ могут быть реализованы в библиотеках DLL. Например некоторые программы могут содержать много различных модулей и при работе использовать только часть из них. Таким образом операционная система и программы загружаются быстрее, работают быстрее и занимают меньше места на диске компьютера.  
**Что такое DLL?**  
      DLL — это библиотека, содержащая код и данные, которые могут использоваться несколько программами одновременно. Например, в операционных системах Windows, библиотека **Comdlg32.dll** выполняет общие функции, связанные с диалоговыми окнами. Таким образом каждая программа может использовать функцию, которая содержится в этой библиотеке для реализации диалогового окна Открыть . Это позволяет повысить уровень повторного использования кода и эффективного использования памяти.  
      С помощью библиотек можно реализовать модульность для программы, в виде отдельных компонентов. Например бухгалтерскую программу можно продать по модулям. Каждый модуль может быть загружен в основной программе во время выполнения установки. Отдельные модули загружается только при запросе функций заложенных в них, поэтому загрузка программы выполняется быстрее.  
      Кроме того обновления легче применить для каждого модуля, не влияя на другие части программы. Например имеется программа по зарплате и надо изменить налоговые ставки за каждый год. Когда эти изменения изолированы в библиотеке, можно применить обновления без необходимости построения или установки программы целиком. Давайте рассмотрим пример создания библиотеки с самыми простыми математическими методами, такие как произведение, деление, сумма и разность.  
      Для начала, создадим новый проект, для этого запустите **Microsoft Visual Studio** и перейдите в меню **Файл -> Создать -> Проект…** или выполните сочетание клавиш **Ctrl+Shift+N**.

[](http://1.bp.blogspot.com/-JT9Sb96uuzk/UOGjdgyT5eI/AAAAAAAACIM/t6i6gOQ8Cpg/s1600/create.png)

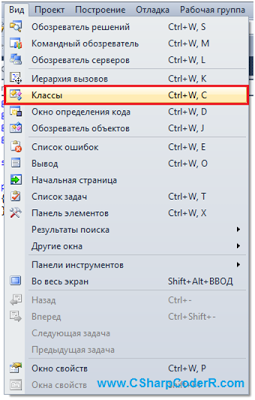
      В открывшемся диалоговом окне выберете версию Framework, а в левой части **«Установленные шаблоны»** выберете **«Visual C#»**, в центральной части вам будет представлен список шаблонов, выберете **«Библиотека классов»** и введите имя библиотеки, можно оставить по умолчанию.

[](http://3.bp.blogspot.com/-R9K3fWrCydo/UOGjtSUe5gI/AAAAAAAACIY/ClM9hVx96QE/s1600/newproject.png)

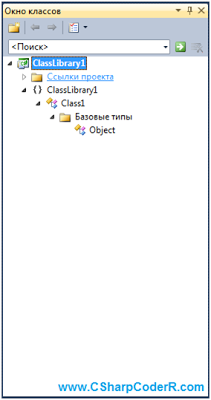
У вас откроется вкладка с классом по умолчанию.

[](http://2.bp.blogspot.com/-9dj_doi0O4I/UOGj81jvdyI/AAAAAAAACIk/_aJ4qoMAz8M/s1600/classlibrary.png)

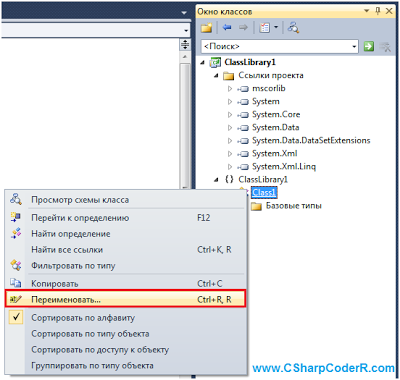
       По умолчанию создается класс **Class1**, переименуем его в класс Calculator. Для этого перейдите в меню Вид - > Классы или выполните сочетание клавиш **Ctrl + W**, с последующем нажатием клавиши **C**.

[](http://3.bp.blogspot.com/-cLCkLFtyXNU/UOGkNMvbCHI/AAAAAAAACIw/G5tq63xNNY8/s1600/class.png)

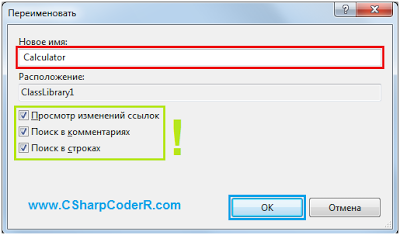
В правой части программы у вас откроется вкладка **«Окно классов»**.

[](http://2.bp.blogspot.com/-_8i0aFdgEGo/UOGkb3WPDWI/AAAAAAAACI8/H8x2Eye4KKE/s1600/pageclass.png)

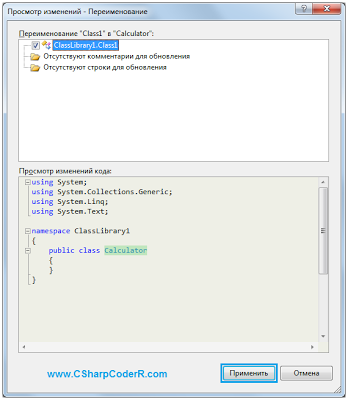
      Выберете по умолчанию созданный класс Class1, сделайте клик правой клавишей мыши по нему и выберете «**Переименовать…**».

[](http://3.bp.blogspot.com/-CbWfnUVD01s/UOGwJEobQgI/AAAAAAAACJ4/UAIDEuSobnk/s1600/rename.png)

      В открывшемся окне введите новое имя класса Calculator и нажмите кнопку OK, обратите внимание, что данное окно позволяет переименовать класс во всем проекте.

[](http://1.bp.blogspot.com/-Kzlbkea0IsU/UOGwcn_fsRI/AAAAAAAACKE/-UixYdJnaas/s1600/renameclass.png)

      После нажатия клавиши OK, вам будет предложено просмотреть список изменений, которые будут внесены в проект, а так же просмотр изменений кода.

[](http://2.bp.blogspot.com/-5JShms1BTLM/UOGw6wHP6YI/AAAAAAAACKQ/F7Cddd0zdw4/s1600/izmeneniya.png)

Добавим в класс **Calculator** несколько методов и добавим к ним описание.

namespace ClassLibrary1

{

///

/// Математический класс

///

public class Calculator

{

///

/// Метод возвращает сумму двух целых чисел

///

/// /// ///

public static int Summ(int firstNumber, int secondNumber)

{

return firstNumber + secondNumber;

}

///

/// Метод возвращает разность двух целых чисел

///

/// /// ///

public static int Division(int firstNumber, int secondNumber)

{

return firstNumber - secondNumber;

}

///

/// Метод возвращает произведение двух чисел

///

/// /// ///

public static long Multiply(long x, long y)

{

return (x \* y);

}

///

/// Метод возвращает деление двух чисел

///

/// /// ///

public static int Residual(int firstNumber, int secondNumber)

{

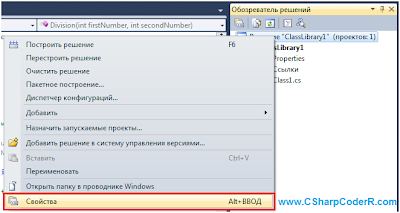
return (firstNumber / secondNumber);

}

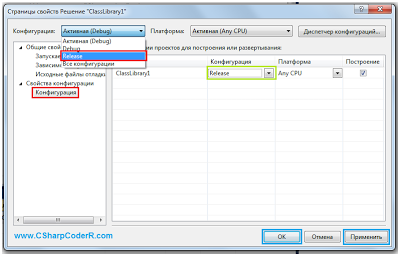
}

}

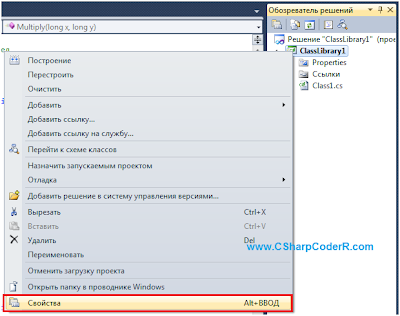
      По умолчанию для всех проектов стоит режим построения **Debug**(режим отладки), переведем проект в режим построения конечной версии (**Release**). Для этого перейдите в обозреватель решений и сделав клик правой клавишей мыши по названию проекта, выберете в открывшемся контекстном меню пункт «**Свойства**».

[](http://3.bp.blogspot.com/-4KJDyt5zs7E/UOMl8rSn3DI/AAAAAAAACLM/p6RajxNkm6c/s1600/property.png)

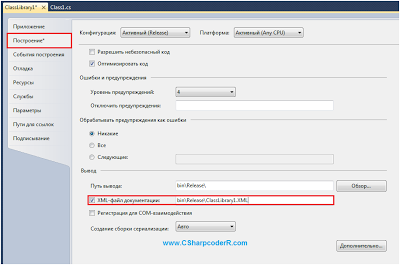
      В открывшемся окне «Страницы свойств Решение “**ClassLibrary1**”» выберете везде конфигурацию Release, как показано на скриншоте ниже.

[](http://3.bp.blogspot.com/-dsPsKGHjq4E/UOMmUrhXukI/AAAAAAAACLY/GAN2Wy0kaHE/s1600/reshenie.png)

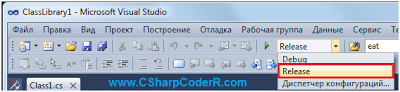
      Наверно все замечали, что при наборе кода, появляется подсказка к методам или функциям. Мы задали такую подсказку в тегах. Но если сейчас просто выполнить построение библиотеки, то при подключении к другим проектам никаких подсказок видно не будет. Что бы устранить данную проблему, нам необходимо сформировать XML файл документации к проекту. Для этого в обозревателе решений выполните клик правой клавишей мыши по названию библиотеки и в открывшемся контекстном меню выберете пункт «**Свойства**».

[](http://2.bp.blogspot.com/-fz7hZcXJo9s/UOMm2HUwCCI/AAAAAAAACLk/rZElOl_be_U/s1600/property1.png)

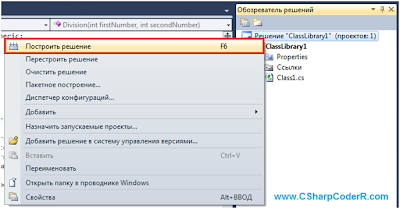
      После этих действий у вас откроется новая вкладка со свойствами проекта **ClassLibrary1**. Выберете в вкладку «**Построение**» и найдите раздел «Вывод», там вам будет предложено ввести путь куда будет выполнено построение конечной версии библиотеки и пункт необходимый нам для построения xml файла документации, тут вам необходимо просто поставить галочку как показано на скриншоте ниже. Тут важно чтобы библиотека и файл документации находились в одном месте, поэтому проверьте чтобы их путь вывода совпадал.

[](http://2.bp.blogspot.com/-qOvL2VLtt1w/UOMnBYXTd5I/AAAAAAAACLw/V6ews0AiwXM/s1600/xml.png)

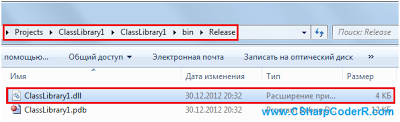
      Остались последние шаги и мы получим готовую для использования библиотеку. И так продолжим, вам необходимо в верхней части программы выбрать режим конфигурации **Release** как показано на скриншоте ниже.

[](http://4.bp.blogspot.com/-pNYfSZeyDl0/UOMnYbPJCAI/AAAAAAAACL8/aHOAdTIpciA/s1600/release.png)

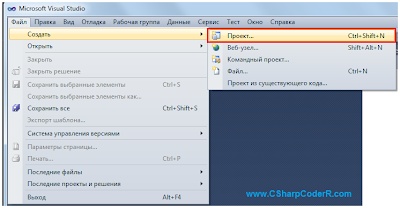
После этого, выполним построение решения. Нажав на клавиатуре клавишу **F6.**

[](http://2.bp.blogspot.com/-ls11ZSd3dnw/UOMnsR6XIsI/AAAAAAAACMM/-DWxyhKsC_0/s1600/postroenie.png)

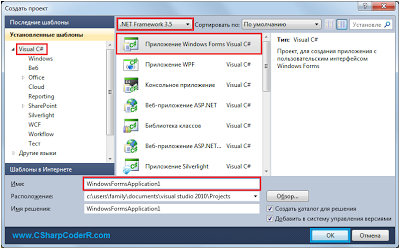
      Как только программа закончит построение, можно перейти в директорию с проектом и посмотреть что получилось. На этом этапе закончилось создание библиотеки.

[](http://3.bp.blogspot.com/-Zi_S4SrUwxg/UOMoAALVqoI/AAAAAAAACMY/JtRF-LeDpSk/s1600/library.png)

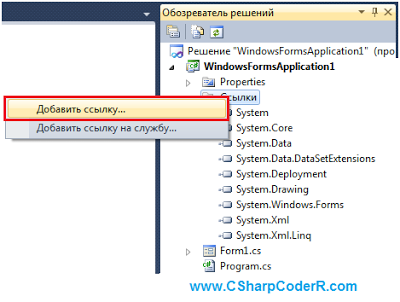
Для проверки работоспособности библиотеки создадим тестовый проект. Выполните **Файл -> Создать -> Проект…**

[](http://1.bp.blogspot.com/-0DC28knNS9g/UOMoNHDnJRI/AAAAAAAACMg/BrPiqbLX1cU/s1600/createproject.png)

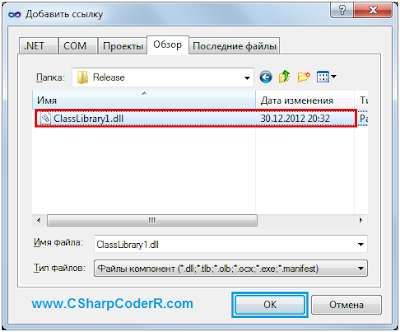
Выберете из предложенных шаблонов, шаблон «Приложение Windows Forms Visual C#». Задайте имя проекта и нажмите кнопку OK.

[](http://1.bp.blogspot.com/-pA982-Cgu3I/UOMogMob_lI/AAAAAAAACMs/csldji-f3N0/s1600/newproject2.png)

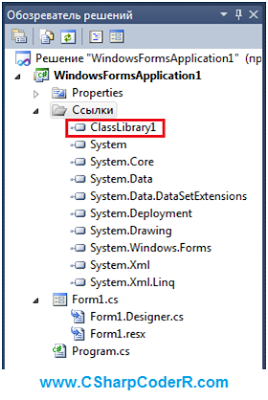
      После создания проекта, в обозревателе решений сделайте клик правой клавишей мыши по разделу «**Ссылки**» и выберете в появившемся контекстном меню пункт «**Добавить ссылку…**».

[](http://1.bp.blogspot.com/-SmOiz2grrdY/UOMo07OIKTI/AAAAAAAACM4/VL4ebELf3WI/s1600/addlibrary.png)

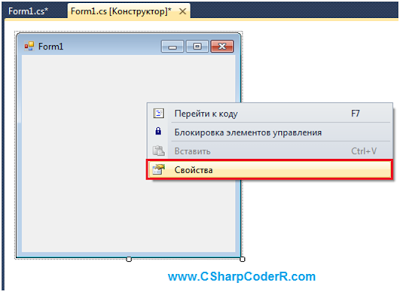
Выберете вкладку «Обзор» и укажите вашу библиотеку.

[](http://3.bp.blogspot.com/-14h0CCFTSDM/UOMpCv1vcII/AAAAAAAACNE/P8G2_yHZAz0/s1600/classlibrary1.png)

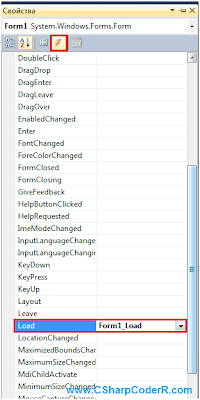
Если вы все успешно выполнили, в разделе «**Ссылки**» у вас появится название вашей библиотеки.

[](http://2.bp.blogspot.com/-SKC8A523Dqg/UOMpSoQq6KI/AAAAAAAACNQ/INfSdSRhdRs/s1600/obozrevatel.png)

      Сделайте клик правой клавишей мыши по главной форме вашего проекта и в открывшемся контекстном меню выберете пункт «**Свойства**».

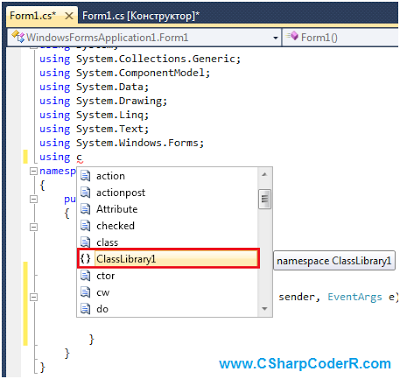
[](http://2.bp.blogspot.com/-0zPeMm9WqYU/UOMpk6gxiLI/AAAAAAAACNc/UGUTLQ92lzI/s1600/property2.png)

В боковой панели откроются «**Свойства**» формы. Найдите **метод Load** и сделайте двойной клик левой клавишей мыши по нему, у вас откроется новая вкладка с добавленным методом **Form1\_Load**.

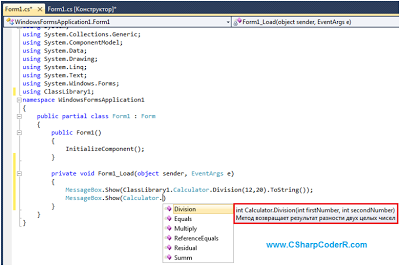
[](http://1.bp.blogspot.com/-NpYc90miDyM/UOMp2G_PEmI/AAAAAAAACNo/T0J-CmOpF94/s1600/form_load.png)

Добавим пространство имен с названием нашей библиотеки.

using ClassLibrary1;

[](http://2.bp.blogspot.com/-ufNAf3uoVlc/UOMqPDh0rZI/AAAAAAAACN4/2YAbe6SPxhg/s1600/classlibrary2.png)

      Пропишите вызов нескольких методов из библиотеки и вывод результата в диалоговом окне «**MessageBox**». Обратите внимание что при выборе методом из библиотеки у вас показывается подсказка которую мы прописывали. Если такого не происходит то, xml файл документации отсутствует в директории с библиотекой. Пример подсказок вы можете посмотреть на скриншоте ниже.

[](http://1.bp.blogspot.com/-jf_QQ6LXvpI/UOMqm5HQ8II/AAAAAAAACOA/QulhvTBkv5k/s1600/description.png)

      Запустите проект, нажав клавишу **F5**. И вы увидите результат выполнения методов прописанных в библиотеке, с параметрами которые вы передали при вызове.

Ссылка для скачивания примера библиотеки: [Яндекс.Диск](https://yadi.sk/d/FiimuKRrfafty)

Ссылка для скачивания примера: [Яндекс.Диск](https://yadi.sk/d/eTe040I7fafux)

yandex