C#10 .NET 6

- versions
    - C# 10 .NET 6 (Dec 2021)
    - C#11 .NET 7 / C# 12 .NET 8.0 (introduction later)
    - **BOOK: C#10 .NET 6**

Advantage: .NET

- *Interoperability with existing code*: This is (of course) a good thing. Existing .NET Framework software can interop with newer .NET Core software, and vice versa, through .NET Standard.
- *Support for numerous programming languages*: .NET Core applications can be created using C#, F#, and VB.NET programming languages (with C# and F# being the primary languages for ASP.NET Core).
- *A common runtime engine shared by all .NET Core languages*: One aspect of this engine is a well-defined set of types that each .NET Core language understands.
- *Language integration*: .NET Core supports cross-language inheritance, cross-language exception handling, and cross-language debugging of code. For example, you can define a base class in C# and extend this type in Visual Basic.
- *A comprehensive base class library*: This library provides thousands of predefined types that allow you to build code libraries, simple terminal applications, graphical desktop applications, and enterprise-level websites.
- *A simplified deployment model*: .NET Core libraries are not registered into the system registry. Furthermore, the .NET Core platform allows multiple versions of the framework as well as applications to exist in harmony on a single machine.
- *Extensive command-line support*: The .NET Core command-line interface (CLI) is a cross-platform tool chain for developing and packaging .NET Core applications. Additional tools can be installed (globally or locally) beyond the standard tools that ship with the .NET Core SDK.

,NET 6:

- Article – MICROSOFT
- 01/27/2022

.NET 6 delivers the final parts of the .NET unification plan that started with .NET 5. .NET 6 unifies the SDK, base libraries, and runtime across mobile, desktop, IoT, and cloud apps. In addition to this unification, the .NET 6 ecosystem offers:

- **Simplified development**: Getting started is easy. New language features in C# 10 reduce the amount of code you need to write. And investments in the web stack and minimal APIs make it easy to quickly write smaller, faster microservices.
- **Better performance**: .NET 6 is the fastest full stack web framework, which lowers compute costs if you're running in the cloud.
- **Ultimate productivity**: .NET 6 and Visual Studio 2022 provide hot reload, new git

tooling, intelligent code editing, robust diagnostics and testing tools, and better team collaboration.

Read more: What's new in .NET 6 | Microsoft Docs

C# 10

What's new in C# 10

- Article - MICROSOFT
- 12/18/2021

C# 10 adds the following features and enhancements to the C# language:

- Record structs
- Improvements of structure types
- Interpolated string handlers
- global using directives
- File-scoped namespace declaration
- Extended property patterns
- Improvements on lambda expressions
- Allow const interpolated strings
- Record types can seal ToString()
- Improved definite assignment
- Allow both assignment and declaration in the same deconstruction
- Allow AsyncMethodBuilder attribute on methods
- CallerArgumentExpression attribute
- Enhanced #line pragma

Read more: What's new in C# 10 - C# Guide | Microsoft Docs

A complete introduction to C# 10 .NET 6 will follow!

Previewing the Building Blocks of the .NET Core Platform (.NET Runtime, CTS, and CLS)

Now that you know some of the major benefits provided by .NET Core, let's preview key (and interrelated) topics that make it all possible: the Core Runtime (formally the CoreCLR and CoreFX), CTS, and the CLS. From a programmer's point of view, .NET Core can be understood as a runtime environment and a comprehensive base class library. The runtime layer contains the set of minimal implementations that are tied specifically to a platform (Windows, iOS, Linux) and architecture (x86, x64, ARM), as well as all of the base types for .NET Core.

Another building block of the .NET Core platform is the *Common Type System*, or *CTS*. The CTS specification fully describes all possible data types and all programming constructs supported by the runtime, specifies how these entities can interact with each other, and details how they are represented in the .NET Core metadata format (more information on metadata later in this chapter; see Chapter 17 for complete details).

Understand that a given .NET Core language might not support every feature defined by the CTS. The *Common Language Specification*, or *CLS*, is a related specification that defines a subset of common types and programming constructs that all .NET Core programming languages can agree on. Thus, if you build .NET Core types that expose only CLS-compliant features, you can rest assured that all .NET Core languages can consume them. Conversely, if you make use of a data type or programming construct that is outside of the bounds of the CLS, you cannot guarantee that every .NET Core programming language can interact

Building blocks in .NET Core (CoreCLR, CoreFX, CTS and CLS)

CoreCLR:

Language Runtime, or CoreCLR, and the .NET Core Libraries, or the CoreFX. The CoreCLR is a set of minimal implementations that are tied specifically to a platform (Windows, iOS, Linux) and architecture (x86, x64, ARM). Only the types that have to be tied to the deployment target are included in the CoreCLR, but are defined as private and should not be accessed through the CoreCLR. The CoreFX is *mostly* platform-

CoreFX:

but are defined as private and should not be accessed through the CoreCLR. The CoreFX is *mostly* platform-agnostic and includes all of the base types for .NET Core. These also include the public implementations of the private types in the CoreCLR. Together, the CoreCLR and CoreFX comprise the .NET Core framework, and must be used together in order to run .NET Core applications.

CTS

Another building block of the .NET Core platform is the *Common Type System*, or *CTS*. The CTS specification fully describes all possible data types and all programming constructs supported by the runtime, specifies how these entities can interact with each other, and details how they are represented in the .NET Core metadata format (more information on metadata later in this chapter; see Chapter 17 for complete details).

BCL

The Role of the Base Class Libraries

The .NET Core platform also provides a set of base class libraries (BCLs) that are available to all .NET Core programming languages. Not only does this base class library encapsulate various primitives such as threads, file input/output (I/O), graphical rendering systems, and interaction with various external hardware devices, but it also provides support for a number of services required by most real-world applications.

The base class libraries define types that can be used to build any type of software application and for components of the application to interact with each other.

C#

What C# Brings to the Table

C# is a programming language whose core syntax looks *very* similar to the syntax of Java. However, calling C# a Java clone is inaccurate. In reality, both C# and Java are members of the C family of programming languages (e.g., C, Objective-C, C++) and, therefore, share a similar syntax.

The truth of the matter is that many of C#'s syntactic constructs are modeled after various aspects of Visual Basic (VB) and C++. For example, like VB, C# supports the notion of class properties (as opposed to traditional getter and setter methods) and optional parameters. Like C++, C# allows you to overload operators, as well as create structures, enumerations, and callback functions (via delegates).

Moreover, as you work through this text, you will quickly see that C# supports a number of features traditionally found in various functional languages (e.g., LISP or Haskell), such as lambda expressions and anonymous types. Furthermore, with the advent of *Language Integrated Query* (LINQ), C# supports a number of constructs that make it quite unique in the programming landscape. Nevertheless, the bulk of C# is indeed influenced by C-based languages.

Because C# is a hybrid of numerous languages, the result is a product that is as syntactically clean as (if not cleaner than) Java, is about as simple as VB, and provides just about as much power and flexibility as C++. Here is a partial list of core C# features that are found in all versions of the language:

C# - developed over many versions

- See the list in the book

Managed vs. Unmanaged Code

It is important to note that the C# language can be used only to build software that is hosted under the .NET Core runtime (you could never use C# to build a native COM server or an unmanaged C/C++-style application). Officially speaking, the term used to describe the code targeting the .NET Core runtime is *managed code*. The binary unit that contains the managed code is termed an *assembly* (more details on assemblies in just a bit). Conversely, code that cannot be directly hosted by the .NET Core runtime is termed *unmanaged code*.

Additional .NET Core Aware Programming Languages

Understand that C# is not the only language that can be used to build .NET Core applications. .NET Core applications can generally be built with C#, Visual Basic, and F#. ASP.NET Core applications are limited to C#.

The .NET framework had a very large list of language implementations. .NET Core, being relatively new, has the three provided by the .NET Core team at Microsoft and (at the time of this writing) one other language, PeachPie, an implementation of PHP for .NET Core.

New Features in C# 9

C# 8, released on November 10, 2020, with .NET 5, adds the following features:

- Records
- Init-only setters
- Top-level statements
- Pattern matching enhancements
- Performance improvements for interop
- "Fit and finish" features
- Support for code generators

New features in C# 9 are indicated as "(New 9)" in their section headings, and updated features are indicated as "(Updated 9)."

Assemblies

Can be an exe or DLL

- CIL
- Metadata
- Manifest

-Structural overview

- CIL (Common Intermediate Language)
- JIT compiler (Just in Time)
- CLR (Common Language Runtime)
- CTS (Common Type System)
- CLS (Common Language Specification)

- BCL / FCL (.NET Base Class Libraries) (Framework Class Library)

About other programming languages –why learn C# .NET?

C/C++

- Manual memory management
- pointers
- C - spaghetti code?

VB

- Have no inheritance
- Is not fully object oriented
- have no parametrized classes

Java

- Java normal used from front to end
- Slooooow
- Hard to integrate with other languages

COM (Microsoft Component Object Model)

- Have no inheritance
- Complex to use

.NET is the solution

- .NET is a new model for software development
    - .NET CORE

.NET IS PLATFORM INDEPENDENT!

Advantages:

- Can be used together with WIN32 code
    - for example COM and raw C
- Total integration
    - Inheritance possible crossing languages
    - Exception handling possible crossing languages
    - Debug possible crossing languages
- Base class libraries prevent raw API call
- Simplified deployment model without use of system registry

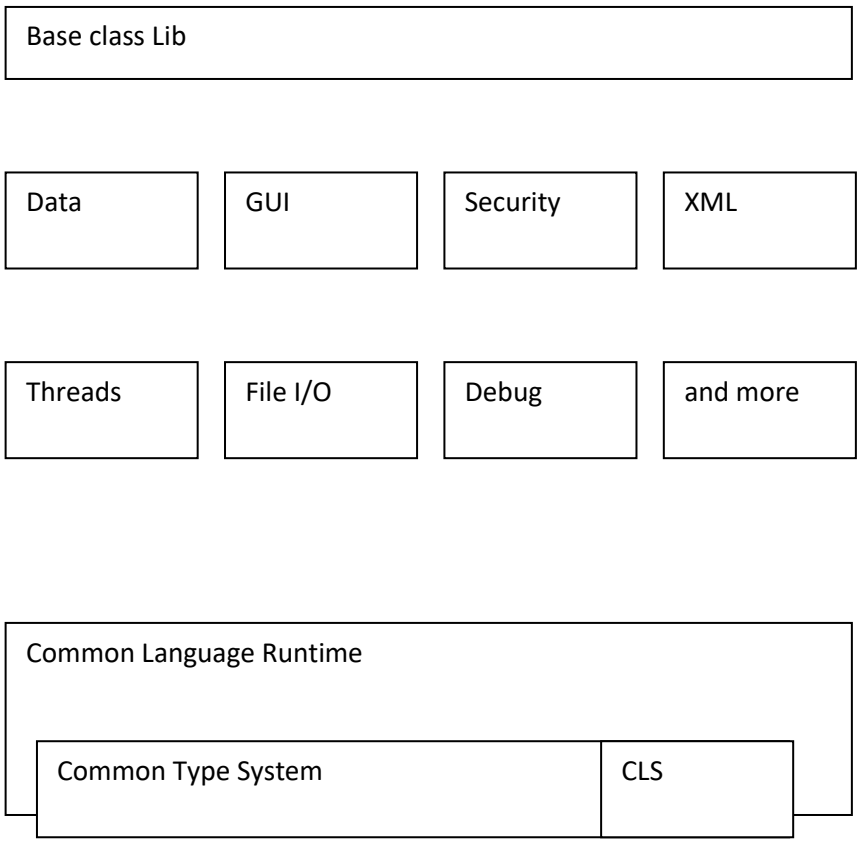
Building blocks: CLR, CTS and CLS

- Runtime of common .NET types (CLR: Common Language Runtime)
- Definition of common types and language constructs:
    - CTS: Common Type System
    - CLS: Common Language Specification

.NET base class library

- Used by all .NET languages
- Encapsulates file handling, threads etc.
- database handling
- XML
- Security
- WEB
- + much more!

Overview - .NET



C# - the best from Java and C++

- Created by Anders Heilsberg (Borland Delphi)

- C# looks very much Java
- Uses a single file for classes(C++ uses .cpp and .h files)
- Java is polished C++
- C# is polished Java
- C# has structs as in C++
- C# has enumerations as in C++
- C# has pointers as in C++ (advanced C# techniques)
- Automatically memory management (as in Java – no use of delete)
- Is (of course) supported by .NET

Other programming languages in the Microsoft suite

- Visual Basic
- J++
- MC++
- JScript .NET
- F#

- Other languages use plug-in's (example: COBOL)

IL: Intermediate Language

- In .NET code is **compiled to IL** (Also called CIL:Common IL) Code
    - Is an "assembly"
        - CIL instructions runs at runtime (Java like)
    - Assemblies contain also **metadata** (information about classes, methods etc.)
    - Assemblies are also described using a **manifest**
- Output is a dll or exe binary file

More about assemblies

- Can be an "one-to-one" relation
    - One assembly to a .dll or .exe
- Can also consist of multi files

    - With a primary module
        - containing assembly manifest
    - Using secondary modules
- Multi files give better flexibility

A module contains logic related work

More about Common Intermediate Language

- CIL is the level next to platform specific instructions

```
.method public instance int32 Add(int32 x,
int32 y) cil managed
{
// Code size 8 (0x8)
.maxstack 2
.locals init (int32 V_0)
IL_0000: ldarg.1
IL_0001: ldarg.2
IL_0002: add.ovf
IL_0003: stloc.0
IL_0004: br.s IL_0006
IL_0006: ldloc.0
```

```
IL_0007: ret
} // end of method Calc::Add
```

Advantages CIL

- More languages can be combined
- platform independent

Type Metadata

- **Assemblies contains also metadata**
- Metadata describes classes ++.
- Thereby the dll/exe becomes self-descriptive and registration in System Registry not used

Assembly manifest

- Contains a list of external used assemblies

From CIL to platform specific instructions

- assemblies contain CIL instructions and metadata
    - is compiled by the JIT compiler to CPU instructions
    - is also called the ”Jitter”
    - .NET runtime has a built-in Jitter

CTS: Common Type System

- Describes a type (class, struct, - -)

CTS class features

- A class is basic in OOP
- CTS allows
    - virtual and abstract members (polymorphy)
    - CTS classes can only inherit one class (multiple inheritance not possible)

Table 1-1. *CTS Class Characteristics*

Class Characteristic	Meaning in Life
Is the class sealed or not?	Sealed classes cannot function as a base class to other classes.
Does the class implement any interfaces?	An interface is a collection of abstract members that provide a contract between the object and object user. The CTS allows a class to implement any number of interfaces.
Is the class abstract or concrete?	Abstract classes cannot be directly instantiated, but are intended to define common behaviors for derived types. Concrete classes can be instantiated directly.
Class Characteristic	**Meaning in Life**
What is the visibility of this class?	Each class must be configured with a visibility keyword such as public or internal. Basically, this controls if the class may be used by external assemblies or only from within the defining assembly.

CTS struct types

- A struct is a ”light-weight class”
    - Can have a constructor (**parameterized**)
    - Can have methods
- CTS structs are a System.ValueType by inheritance
- CTS structs are **sealed** – cannot be a base class
    - Can implement interfaces

CTS Interface type

- Are abstract definitions, that can be implemented
- has itself no implementation

- Does not inherit a base type (not even System.Object)

CTS Enumeration type

- Grouping values using a specific name

```
public enum PlayerType { Wizard=100, Fighter=200};
```

- Can contain a System.Int32

CTS Delegate type

- Correspond to function pointer in C
- Inherits from System.MulticastDelegate
- A delegate can call another entity

Information about CTS Type members

- CTS contains information about the type
    - visibility?
    - is the type abstract?
    - is the type virtual (can it be overwritten?)
    - is the type static (class level)
    - is the type instance (object level)

See table 1-1 above

CTS Data types

Table 1-2. *The Intrinsic CTS Data Types*

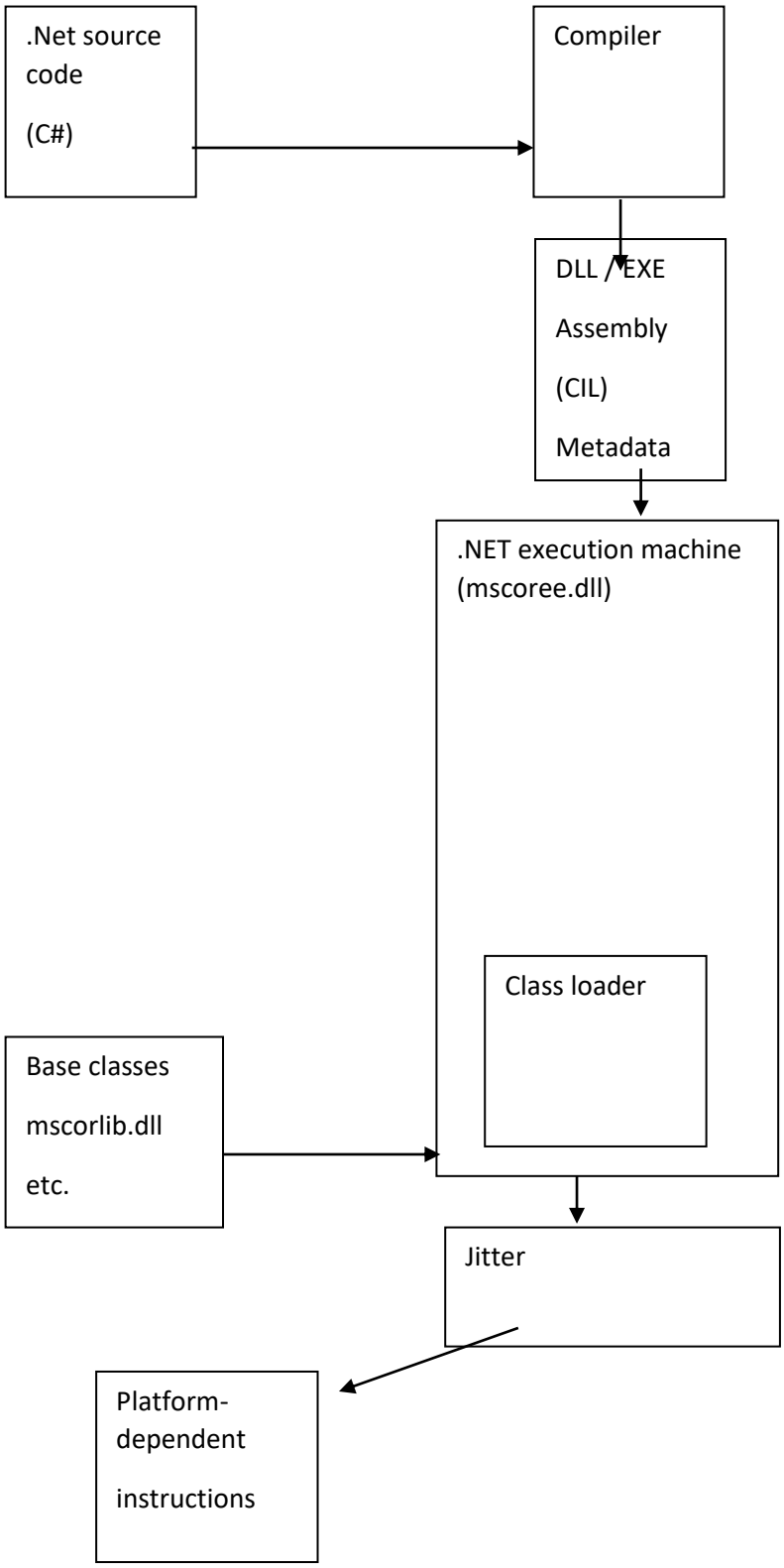
CTS Data Type	VB Keyword	C# Keyword	C++/CLI Keyword
System.Byte	Byte	byte	unsigned char
System.SByte	SByte	sbyte	signed char
System.Int16	Short	short	short
System.Int32	Integer	int	int `or` long
System.Int64	Long	long	__int64
System.UInt16	UShort	ushort	unsigned short
System.UInt32	UInteger	uint	unsigned int `or` unsigned long
System.UInt64	ULong	ulong	unsigned __int64
System.Single	Single	float	float
System.Double	Double	double	double
System.Object	Object	object	object^
System.Char	Char	char	wchar_t
System.String	String	string	String^
System.Decimal	Decimal	decimal	Decimal

CLS: Common Language Specification

- Programming constructs is different in different languages:
    - C# string concatenation uses + operator
    - VB uses & (ampersand)

- CLS is a set of rules the specific compiler must apply to

CLR: Common Language Runtime



.NET namespaces

- Many languages uses built-in class libraries
    - Example: MFC
- C# has not its own class library
    - but uses .NET framework

.NET uses namespaces!

- namespaces also prevent name clashes

The System namespace

```
using System;
public class Hejsa
{
  public static void Main()

  {
      Console.WriteLine("Hallo from C#");
```

```
    }


}
```

- Console is defined in System

List (sample) of namespaces:

Table 1-3. A Sampling of .NET Namespaces

.NET Namespace	Meaning in Life
`System`	Within `System`, you find numerous useful types dealing with intrinsic data, mathematical computations, random number generation, environment variables, and garbage collection, as well as a number of commonly used exceptions and attributes.
`System.Collections System.Collections.Generic`	These namespaces define a number of stock container types, as well as base types and interfaces that allow you to build customized collections.
`System.Data System.Data.Common System.Data.SqlClient`	These namespaces are used for interacting with relational databases using ADO.NET.
`System.IO System.IO.Compression System.IO.Ports`	These namespaces define numerous types used to work with file I/O, compression of data, and port manipulation.
`System.Reflection System.Reflection.Emit`	These namespaces define types that support runtime type discovery as well as dynamic creation of types.
`System.Runtime.InteropServices`	This namespace provides facilities to allow .NET types to interact with unmanaged code (e.g., C-based DLLs and COM servers) and vice versa.
`System.Drawing System.Windows.Forms`	These namespaces define types used to build desktop applications using .NET's original UI toolkit (Windows Forms).
`System.Windows System.Windows.Controls System.Windows.Shapes`	The `System.Windows` namespace is the root for several namespaces that are used in Windows Presentation Foundation applications.
`System.Windows.FormsSystem.Drawing`	The `System.Windows.Forms` namespace is the root for several namespaces used in Windows Forms applications.
`System.Linq System.Linq.Expressions`	These namespaces define types used when programming against the LINQ API.
`System.AspNetCore`	This is one of many namespaces that allows you to build ASP.NET Core web applications and RESTful services.
`System.Threading System.Threading.Tasks`	These namespaces define numerous types to build multithreaded applications that can distribute workloads across multiple CPUs.
`System.Security`	Security is an integrated aspect of the .NET universe. In the security-centric namespaces, you find numerous types dealing with permissions, cryptography, etc.
`System.Xml`	The XML-centric namespaces contain numerous types used to interact with XML data.

- System is a kind of **root**

Referencing External Assemblies

Prior version of the .NET Framework used a common installation location for framework libraries known as the Global Assembly Cache (GAC). Instead of having a single installation location, .NET Core does not use the GAC. Instead, each version (including minor releases) is installed in its own location (by version) on the computer. When using Windows, each version of the runtime and SDK gets installed into c:\Program Files\dotnet.

Adding in additional assemblies into *most* .NET Core projects is done by adding NuGet packages (covered later in this text). However, .NET Core applications targeting (and being developed on) Windows still have access to COM libraries. This will also be covered later in this text.

In order for an assembly to have access to another assembly that you are building (or someone built for you), you need to add a reference from your assembly to the other assembly and have physical access to the other assembly. Depending on the development tool you are using to build your .NET applications, you will have various ways to inform the compiler which assemblies you want to include during the compilation cycle.

CLS – Rules

- CLS must be compliant externally :

Example:

```
class Calc
{
// Exposed unsigned data is not CLS compliant!
public ulong Add(ulong x, ulong y)
{
return x + y;
}
}
```

```
class Calc
{
```

```
public int Add(int x, int y)
{
// As this ulong variable is only used internally,
// we are still CLS compliant.
ulong temp = 0;
...
return x + y;
}
```

Using namespaces

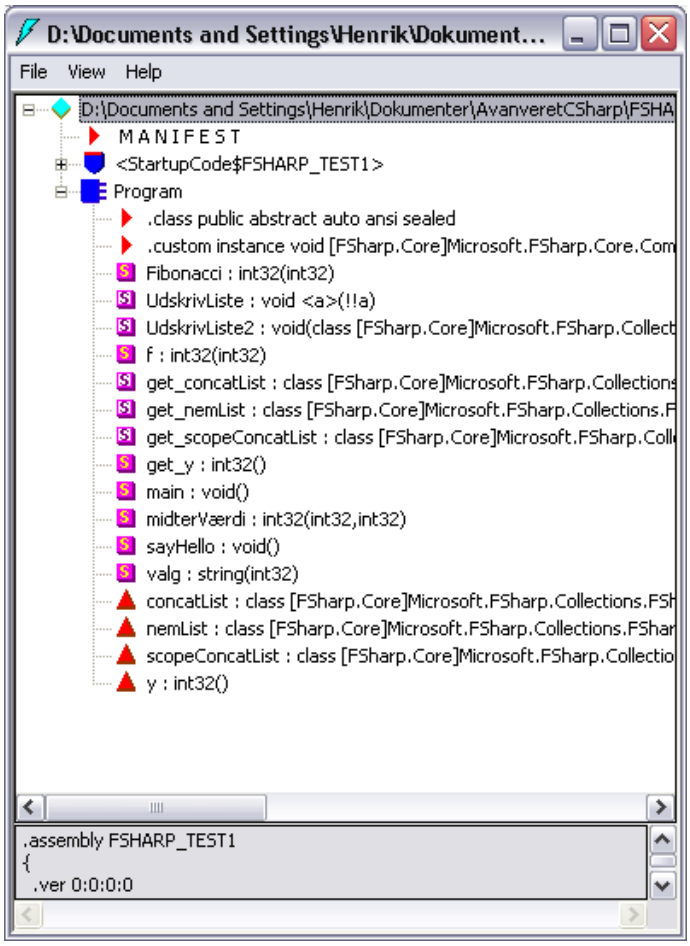
- explicit list
    - example:
    - `using System.Drawing;`
- Full name:
    - example:
    - `System.Drawing.Bitmap bm = new System.Drawing.Bitmap(5,5,);`

More about using extern assemblies

- Cores .NET namespaces exists in mscorlib.dll
- But: The CIL definitions of System.Drawing exists in Syste.Drawing.dll
    - The compiler must have access to this DLL

The dissassembler (ildasm.exe)

- Can read the content of an .NET assembly

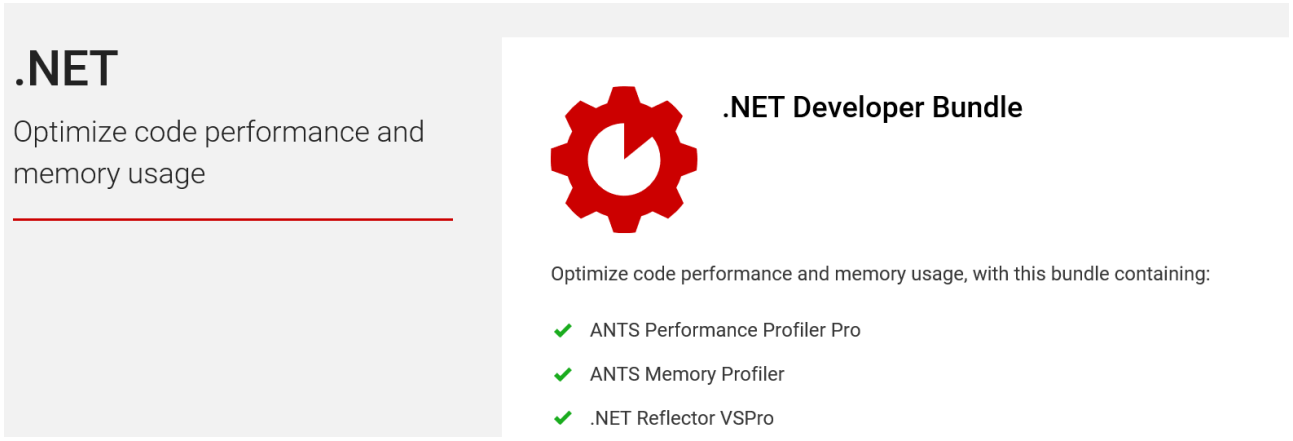


- CIL can be read – can be dumped to a file
- Metadata can be read

Note The `ildasm.exe` program no longer ships with the .NET Core runtime, but must be compiled from the CoreCLR source, which can be located at `https://github.com/dotnet/coreclr` (.NET Core 3.1) or `https://github.com/dotnet/runtime` (.NET 5). The home page for the repo has platform-specific instructions to build and locate the ildasm tool. I have included `ildasm.exe` in the Chapter 1 folder of this book's GIT repo. The good news is that the previous versions of `ildasm.exe` will typically work for .NET Core assemblies developed on Windows.

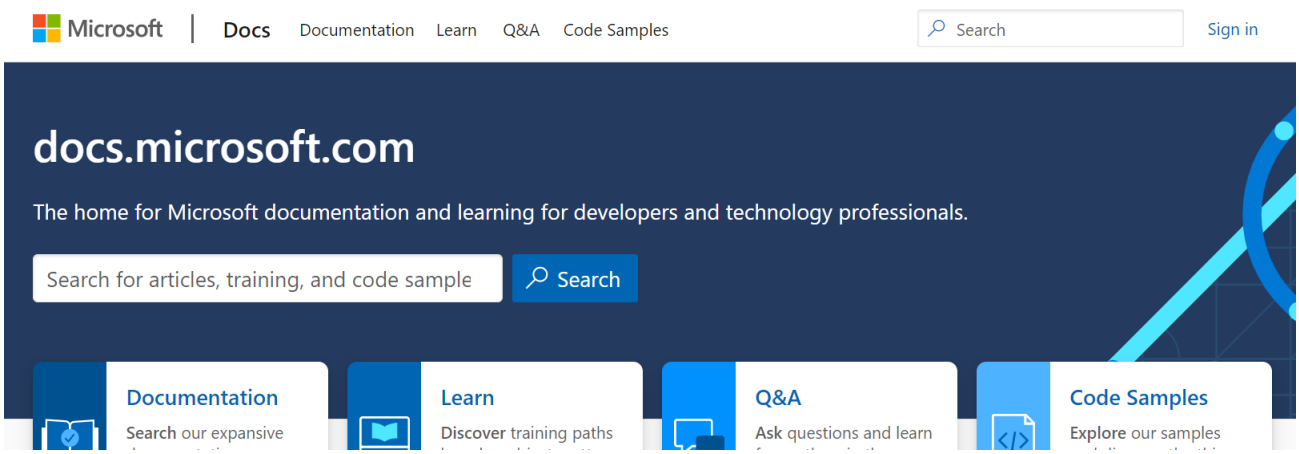
Reflector

- Another object ”browser” (see: www.red-gate.com)



WEB help –

[Developer tools, technical documentation and coding examples | Microsoft Docs]

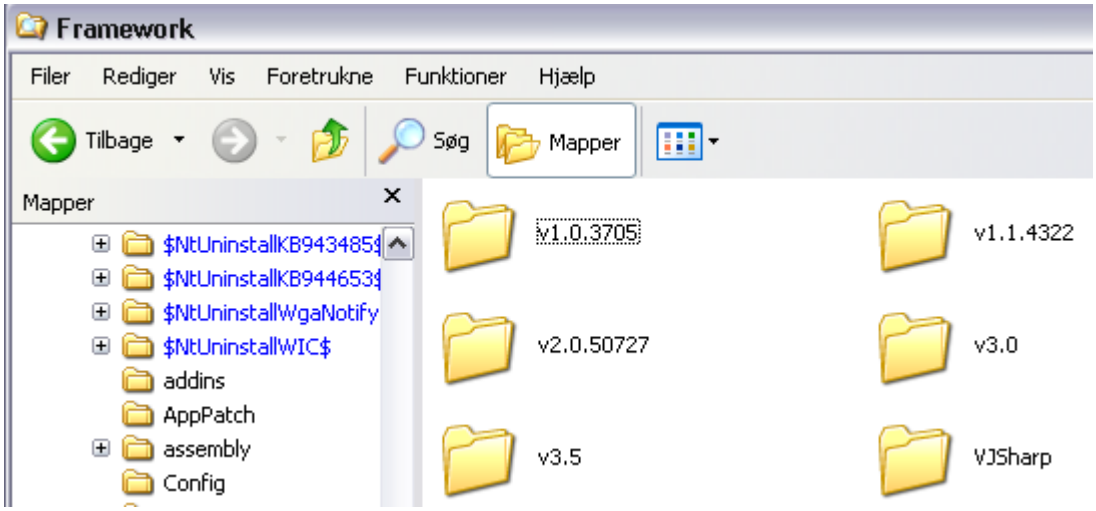


Open source .NET distributions

- http://www.dotgnu.org

More versions of .NET

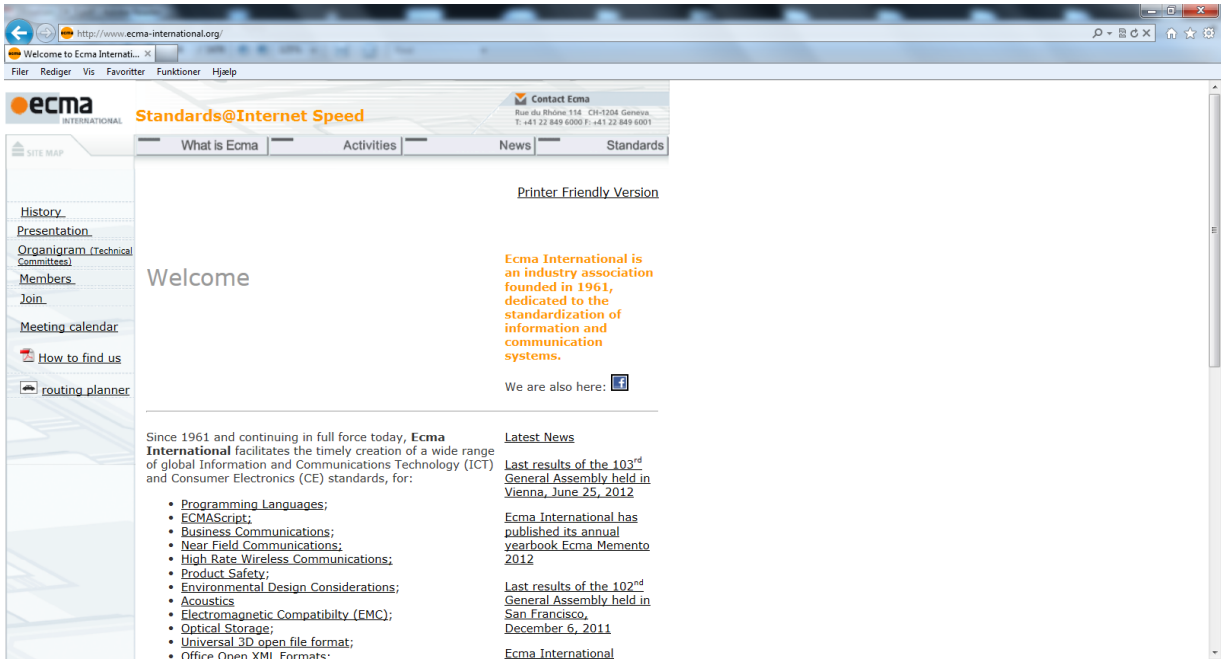
- Can ”co-exist”



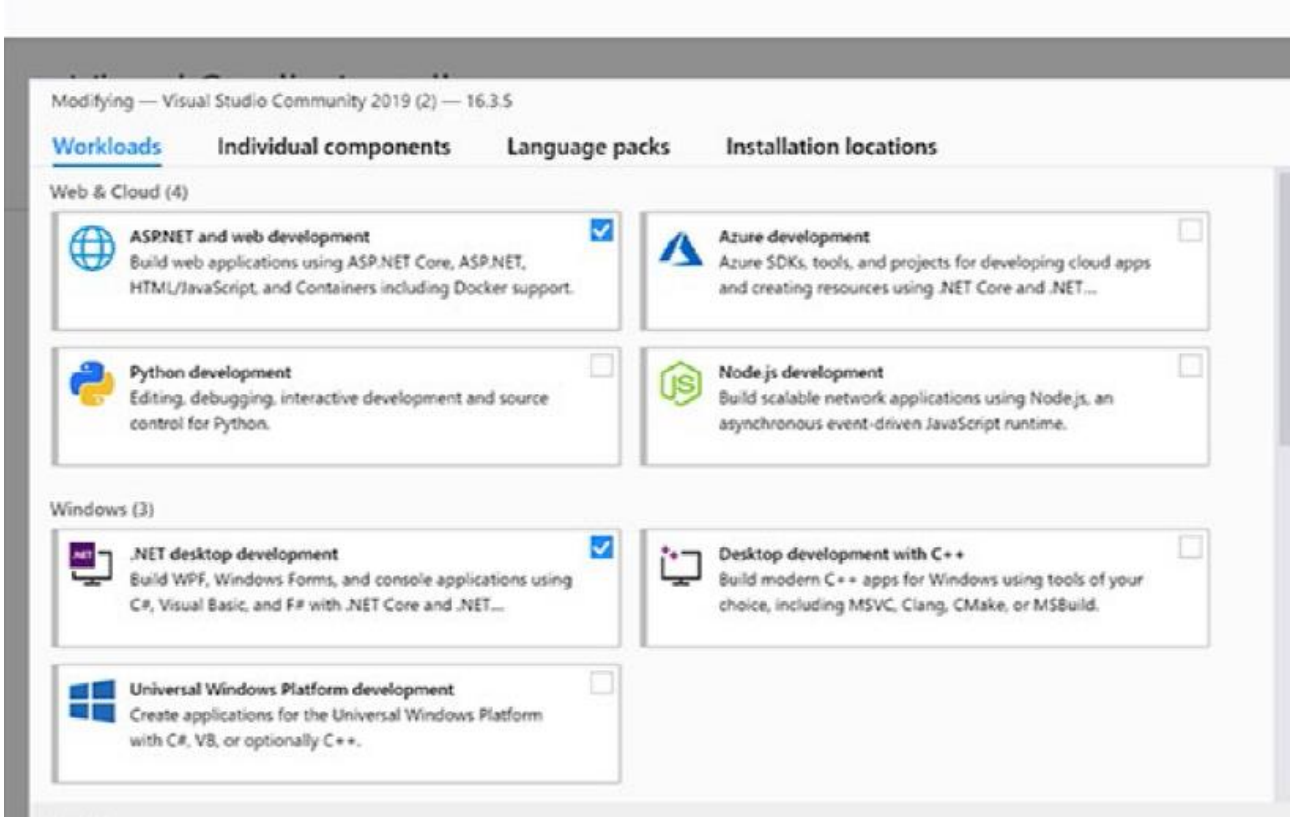
- Remember: The C# Compiler is a part of the .NET API – **NOT Visual Studio**

Platform Independent

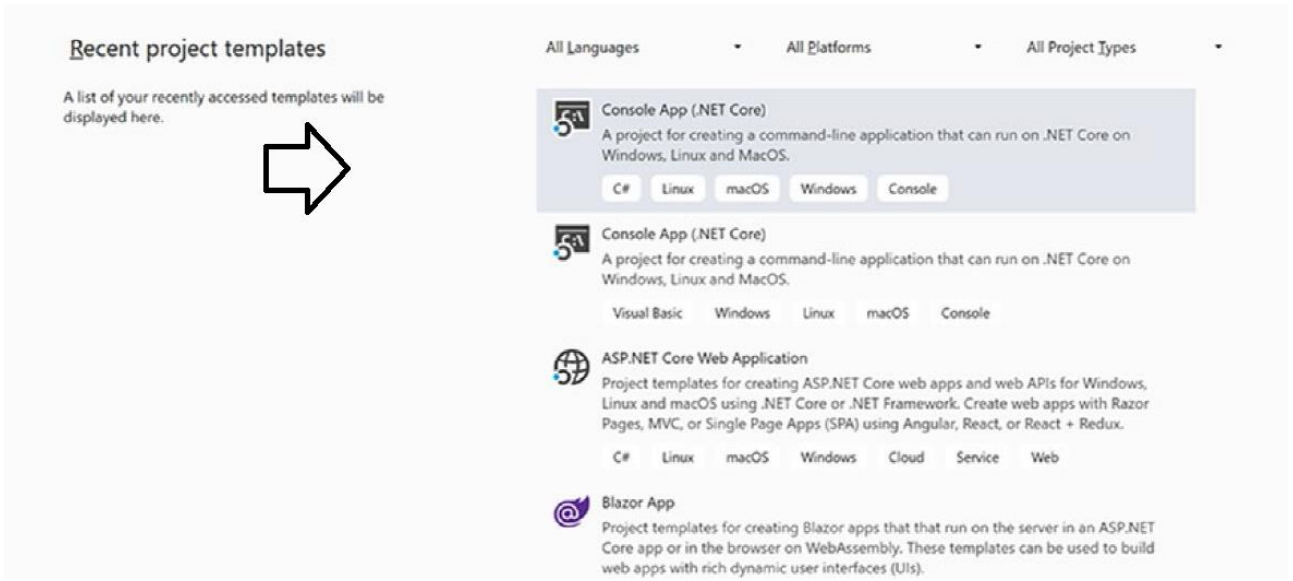
- C# and .NET is described in:
    - ECMA-334 (C#)
    - ECMA-335 (CLI)
-



Install VS



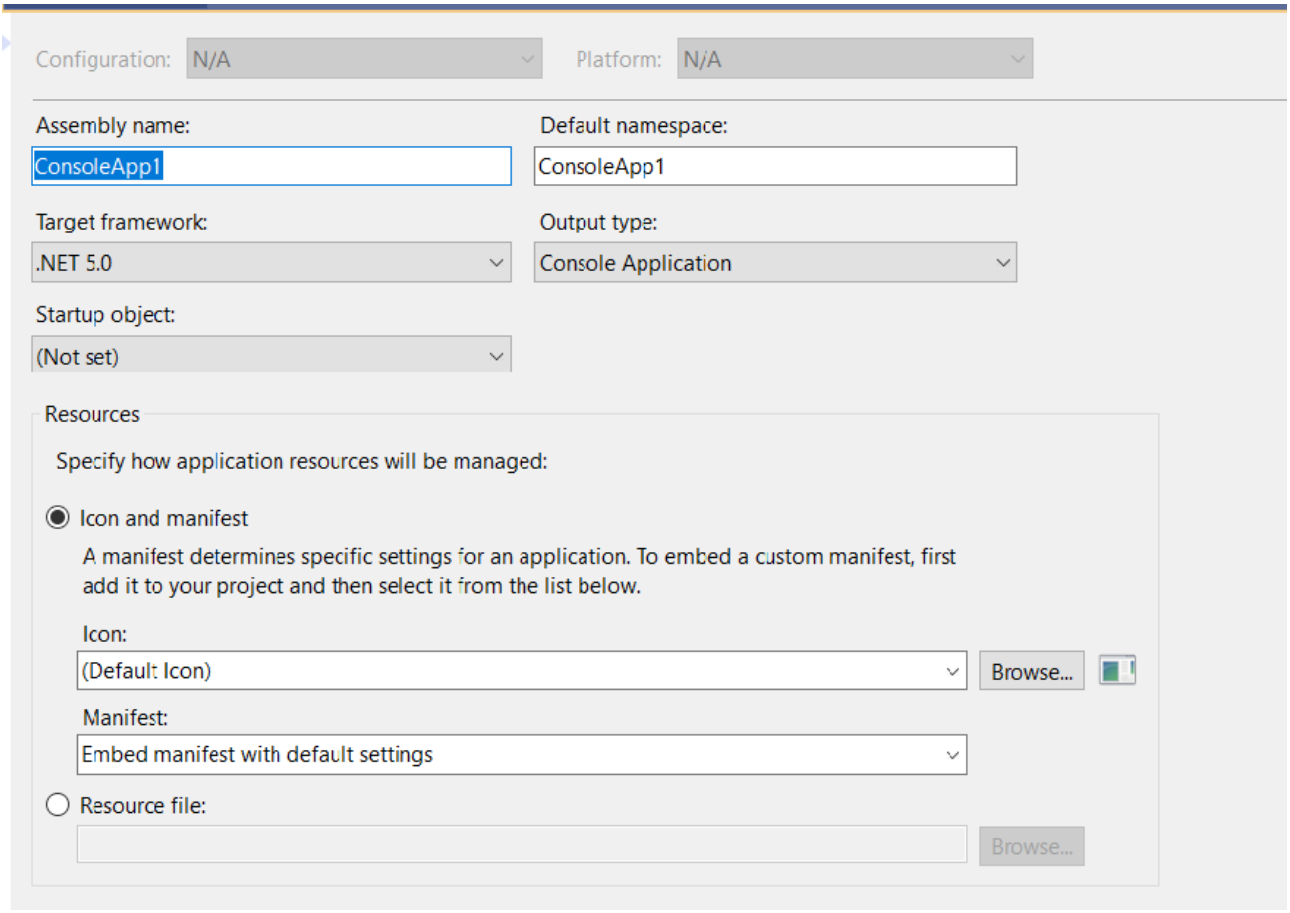
Project type



Installing .NET 5

To get started developing applications with C# 9 and .NET 5 (on Windows, macOS, or Linux), the .NET 5 SDK needs to be installed (which also installs the .NET 5 runtime). All of the installs for .NET and .NET Core are located at the convenient `www.dot.net`. On the home page, click Download and then click "All .NET downloads" under .NET. After clicking "All .NET downloads," you will see the two LTS versions of .NET Core (2.1 and 3.1) and a link for .NET 5.0. Click ".NET 5.0 (recommended)." Once on that page, select the correct .NET 5 SDK for your operating system. For this book, you will need to install the SDK for .NET Core version 5.0.100 or higher, which also installs the .NET, ASP.NET, and .NET Desktop (on Windows) runtimes.

Note The download page has changed since the release of .NET 5. There are now three columns with the headers ".NET," ".NET Core," and ".NET Framework." Clicking "All .NET Core downloads" under the .NET or .NET Core header takes you to the same page. Installing Visual Studio 2019 also installs the .NET Core SDK and runtimes.



OR: Edit Project File

ConsoleApp1.csproj - Notepad

File Edit Format View Help

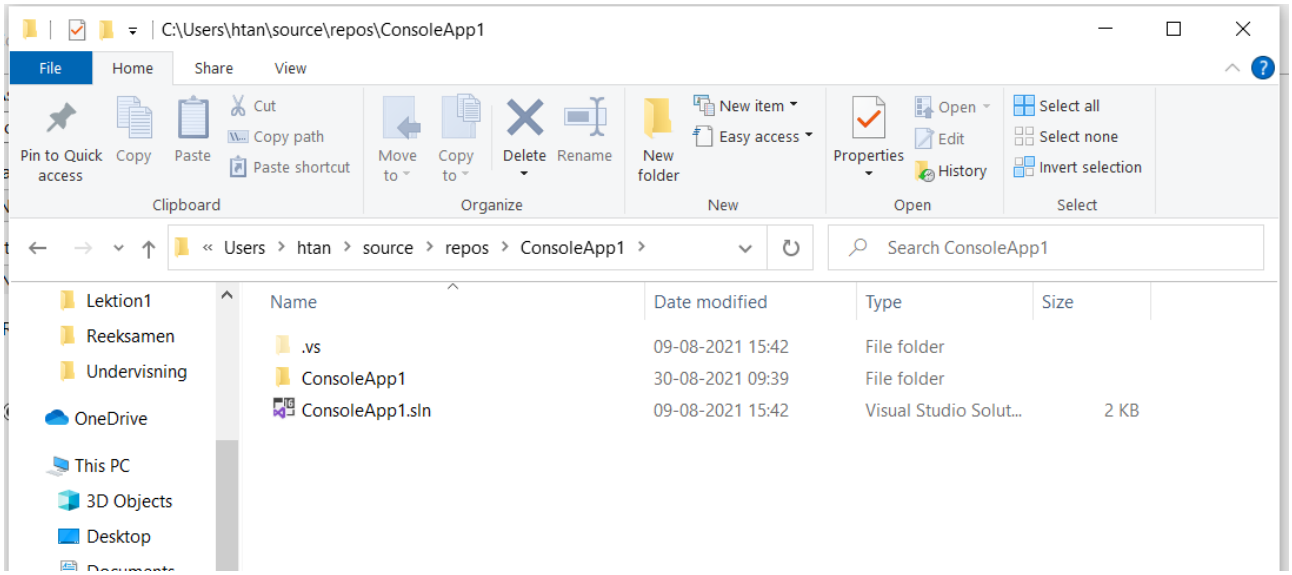
```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>

</Project>
```

To change to a different .NET Core version (such as .NET Core 2.1), simply change the TargetFramework value. You can also change the target framework by right-clicking the project name in solution explorer and selecting Properties.

Project location



Verions and target Framework

Confirming the .NET 5 Install

To confirm the installation of the SDK and the runtimes, open a command window and use the .NET 5 command-line interface (CLI), `dotnet.exe`. The CLI has SDK options and commands available. The commands include creating, building, running, and publishing projects and solutions, and you will see examples of those commands later in this text. In this section, we will examine the SDK options, of which there are four, as shown in Table 2-1.

Table 2-1. *.NET 5 CLI SDK Options*

Option	Meaning in Life
`--version`	Display the .NET SDK version in use
`--info`	Display .NET information
`--list-runtimes`	Display the installed runtimes
`--list-sdks`	Display the installed SDKs
`--version`	Display the .NET SDK version in use

The `--version` option displays the highest version of the SDK installed on your machine, or the version specified in a `global.json` located at or above your current directory. Check the current version of the .NET 5 SDK installed on your machine, enter the following:

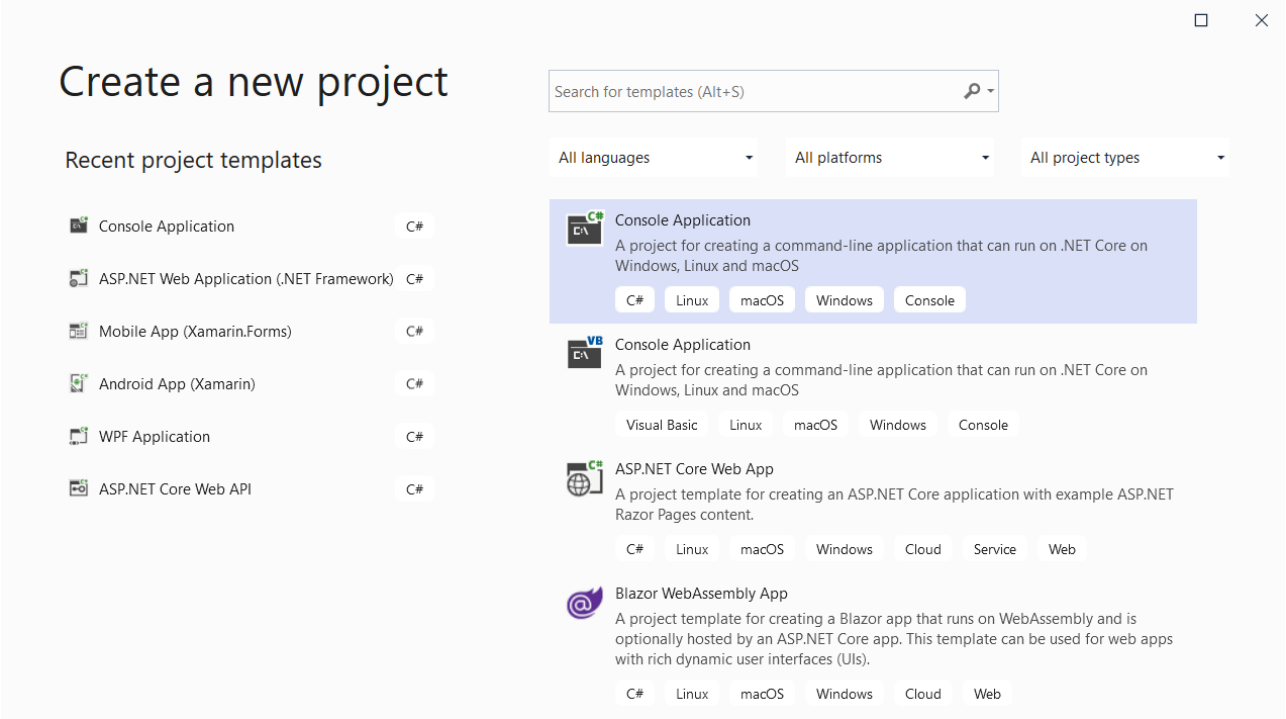
```
dotnet --version
```

For .NET 5.0 projects, the language version is locked into C# 9. Table 2-2 lists the target frameworks (.NET Core, .NET Standard, and .NET Framework) and the default C# version utilized.

Table 2-2. *C# 8 Version and Target Framework*

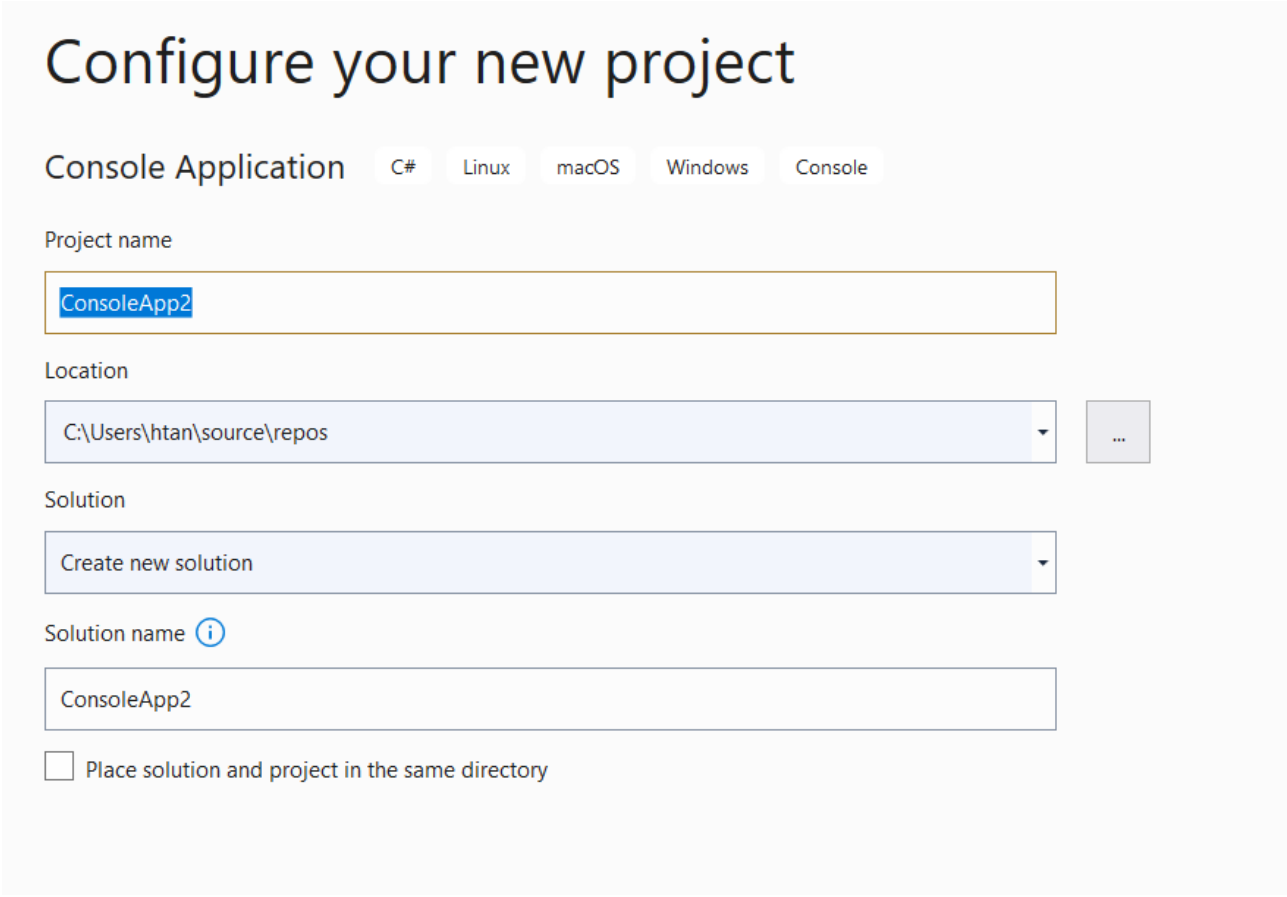
Target Framework	Version	C# Language Version Default
.NET	5.x	C# 9.0
.NET Core	3.x	C# 8.0
.NET Core	2.x	C# 7.3
.NET Standard	2.1	C# 8.0
.NET Standard	2.0	C# 7.3
.NET Standard	1.x	C# 7.3
.NET Framework	all	C# 7.3

Visual Studio 2022

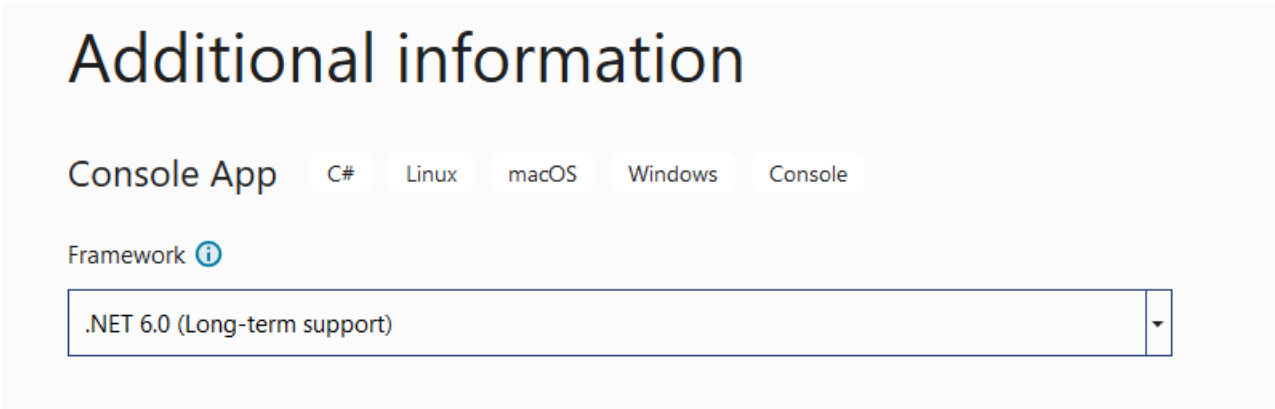


Remember: C# Version!!!

Configure



Choose Target Framework:



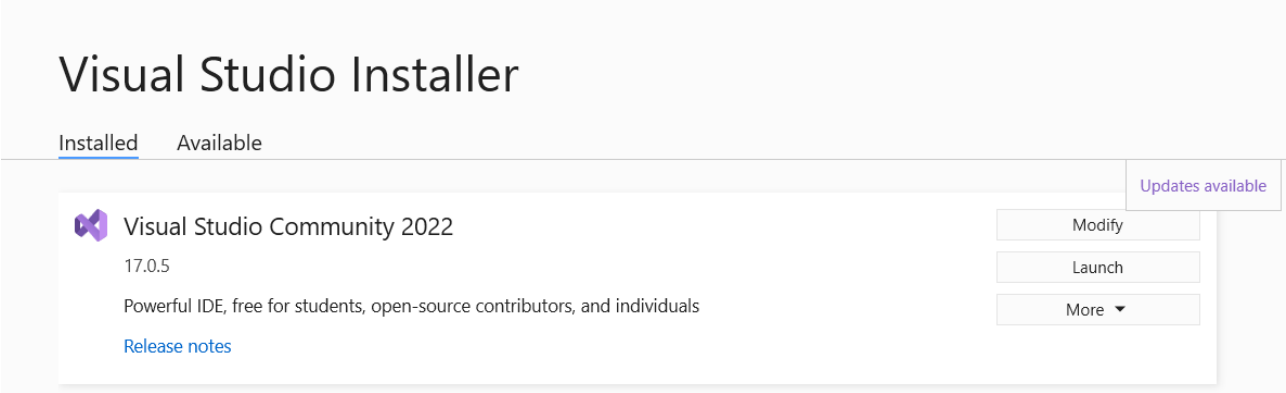
Class diagram??

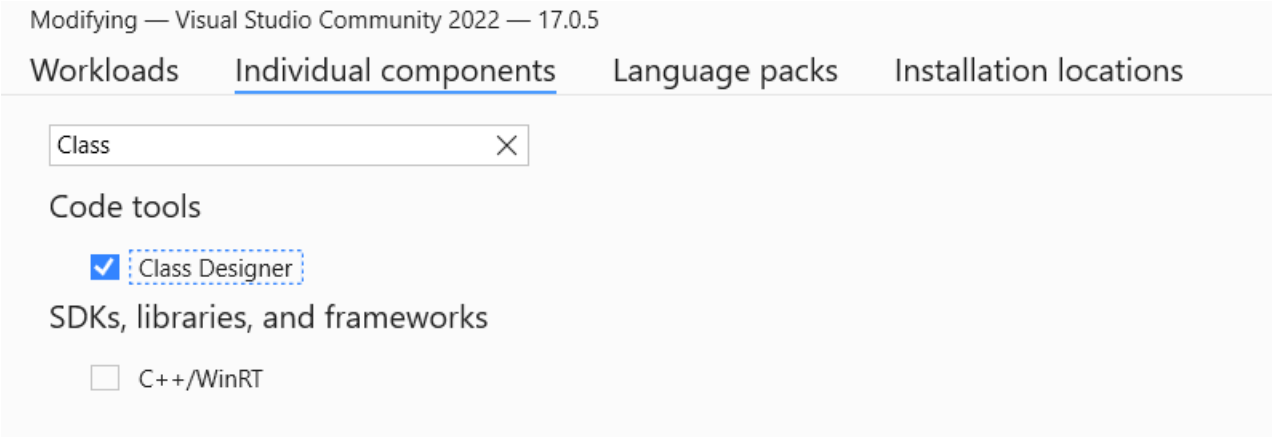
Read more: [Add Class Diagrams to projects (Class Designer) - Visual Studio (Windows) | Microsoft Docs]

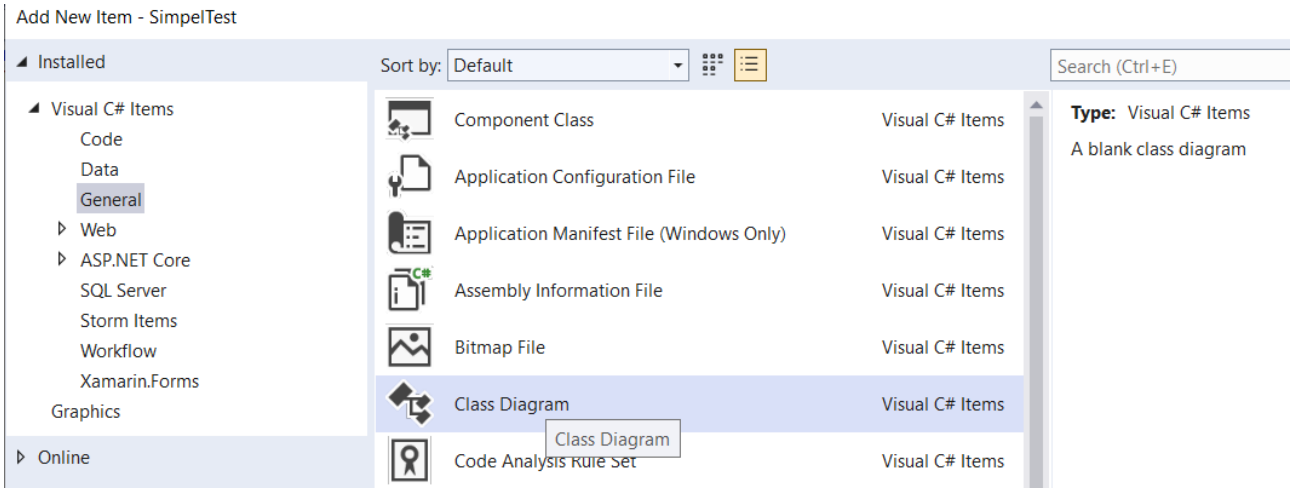
1. Open **Visual Studio Installer** from the Windows Start menu, or by selecting **Tools** > **Get Tools and Features** from the menu bar in Visual Studio.

Visual Studio Installer opens.

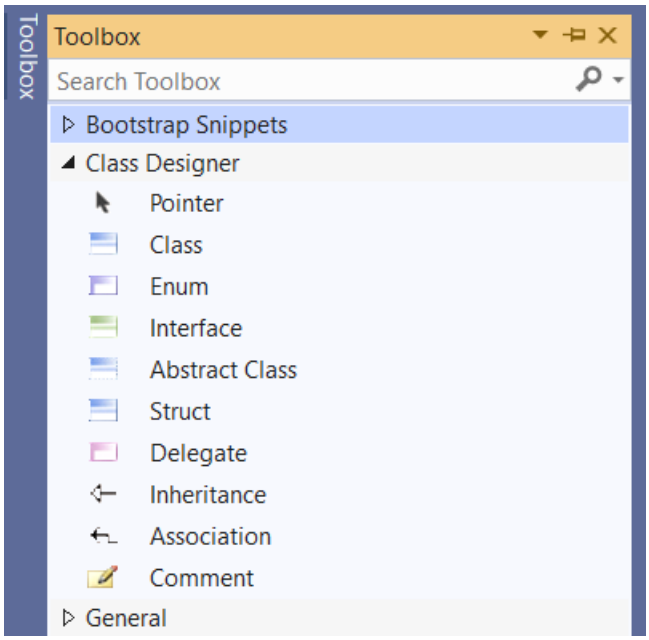
2. Select the **Individual components** tab, and then scroll down to the **Code tools** category.
3. Select **Class Designer** and then select **Modify**.






Add New Item - SimpelTest
Installed
Visual C# Items
Code
Data
General
Web
ASP.NET Core
SQL Server
Storm Items
Workflow
Xamarin.Forms
Graphics
Online
Sort by: Default
Search (Ctrl+E)
Component Class
Application Configuration File
Application Manifest File (Windows Only)
Assembly Information File
Bitmap File
Class Diagram
Code Analysis Rule Set
Visual C# Items
Class Diagram
Type: Visual C# Items
A blank class diagram

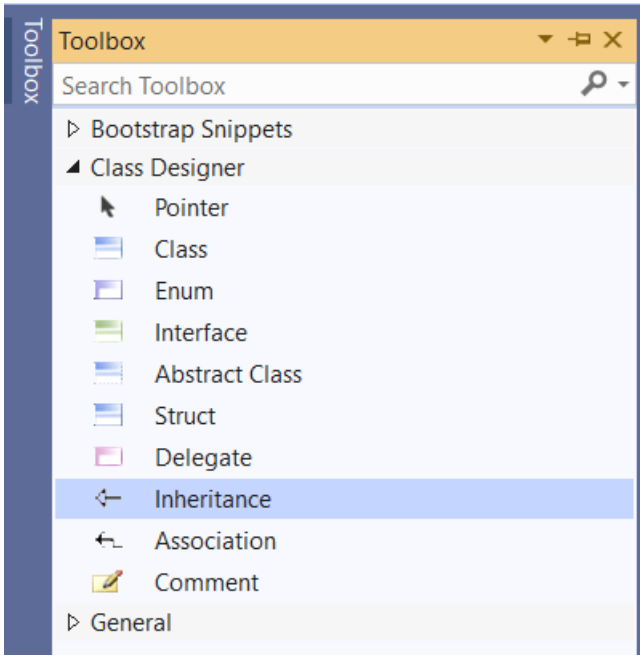
Toolbox

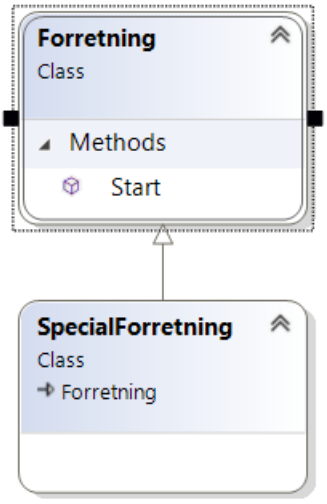


Code

```
1   using System;
2   using System.Collections.Generic;
3   using System.Text;
4
5   namespace ConsoleApp22
6   {
7       public class Forretning
8       {
9           public void Start()
10          {
11              throw new System.NotImplementedException();
12          }
13      }
14  }
```

Inheritance


Toolbox
Toolbox
Search Toolbox
Bootstrap Snippets
Class Designer
Pointer
Class
Enum
Interface
Abstract Class
Struct
Delegate
Inheritance
Association
Comment
General


Forretning
Class
Methods
Start
SpecialForretning
Class
Forretning

```
1   using System;
2   using System.Collections.Generic;
3   using System.Text;
4
5   namespace ConsoleApp22
6   {
7       public class SpecialForretning : Forretning
8       {
9       }
10  }
```

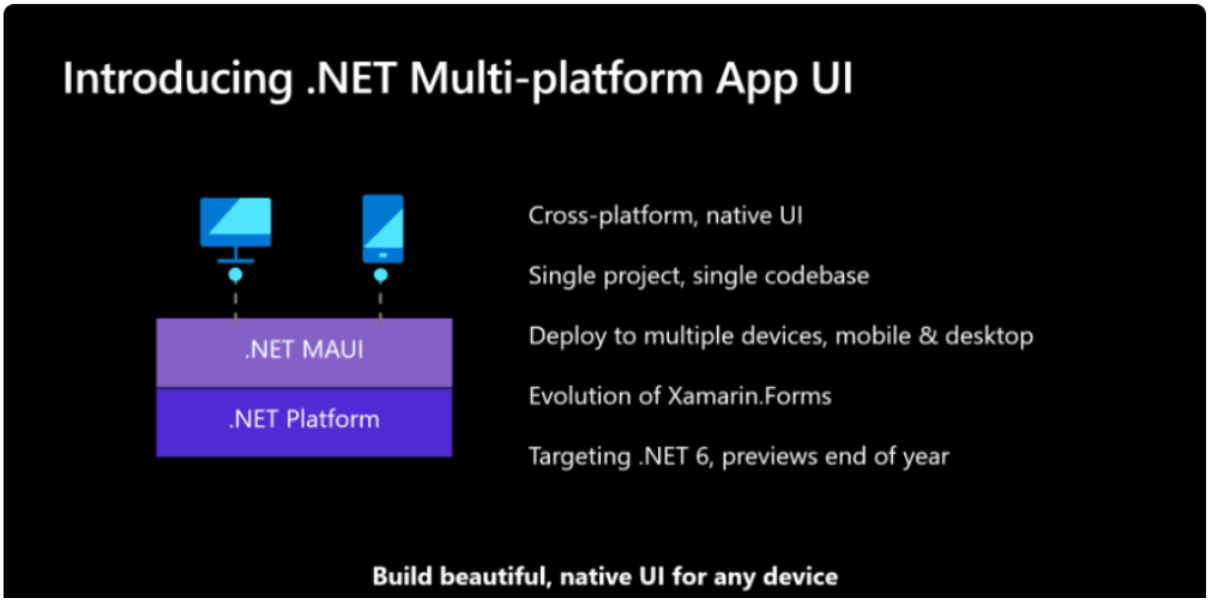
Visual Studio .NET 2022

- Features

- Visual XML editors/designers
- Designer support for Windows Workflow Foundation projects
- Integrated support for code refactoring
- Visual class designer tools
- Plus

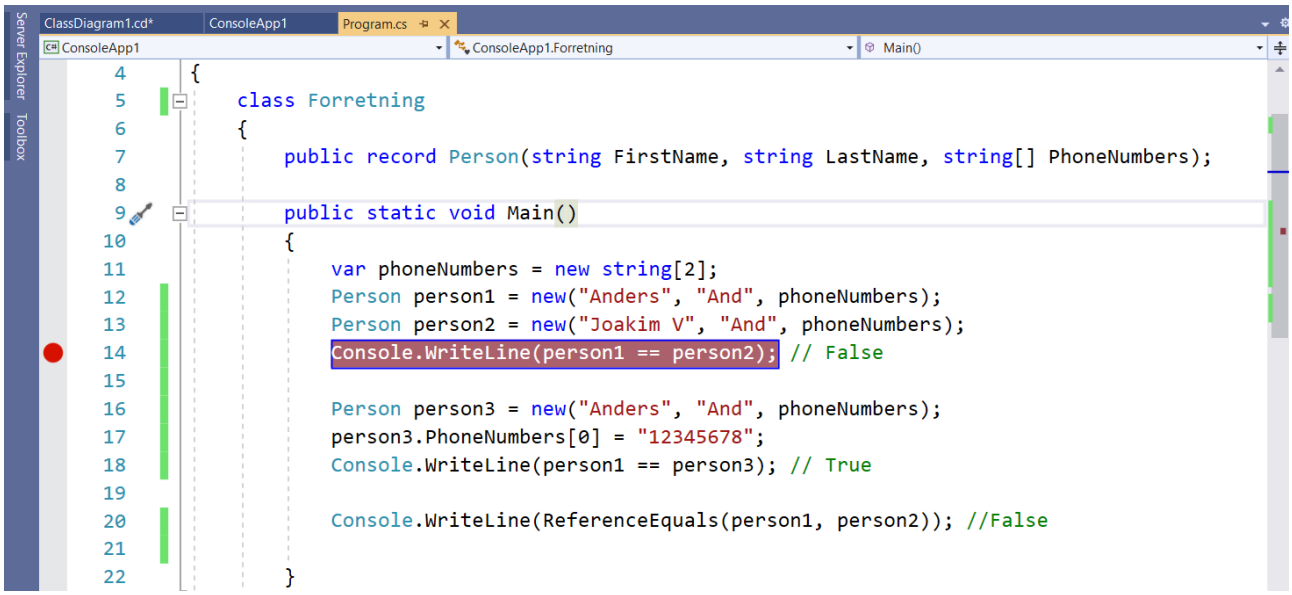
.NET MAUI

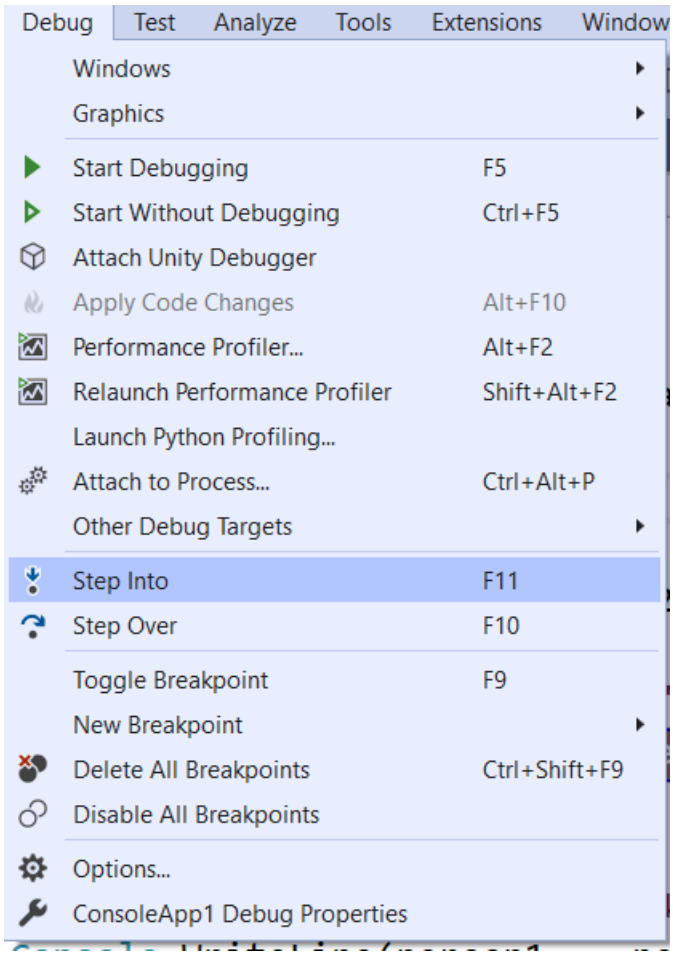
The most remarkable functionality of this new version of the framework will be **.NET MAUI.** Is a technology that allows you to **create** common interfaces for any environment, be it **desktop**, **mobile** or **web**, and on **any operating system**. .NET MAUI is the replacement for *Xamarin Forms* and will add the ability to create desktop applications using *Blazor*.



Debug

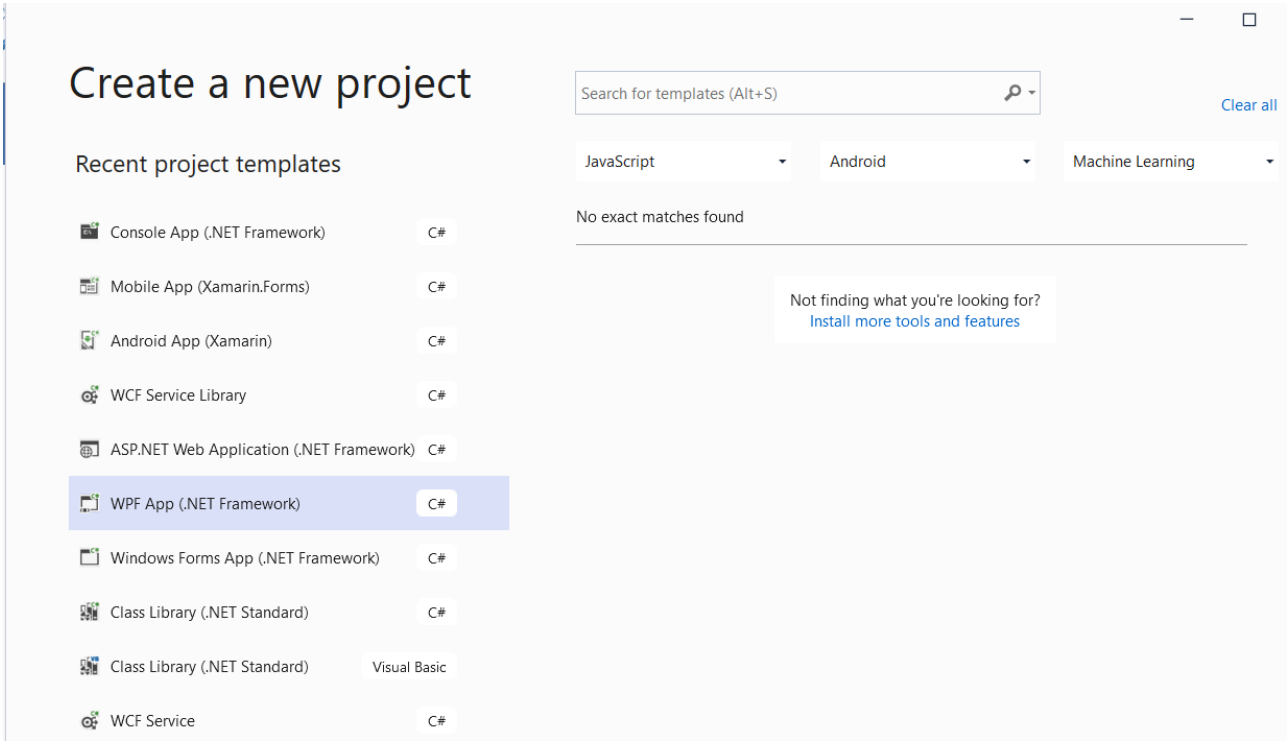
- Breakpoint



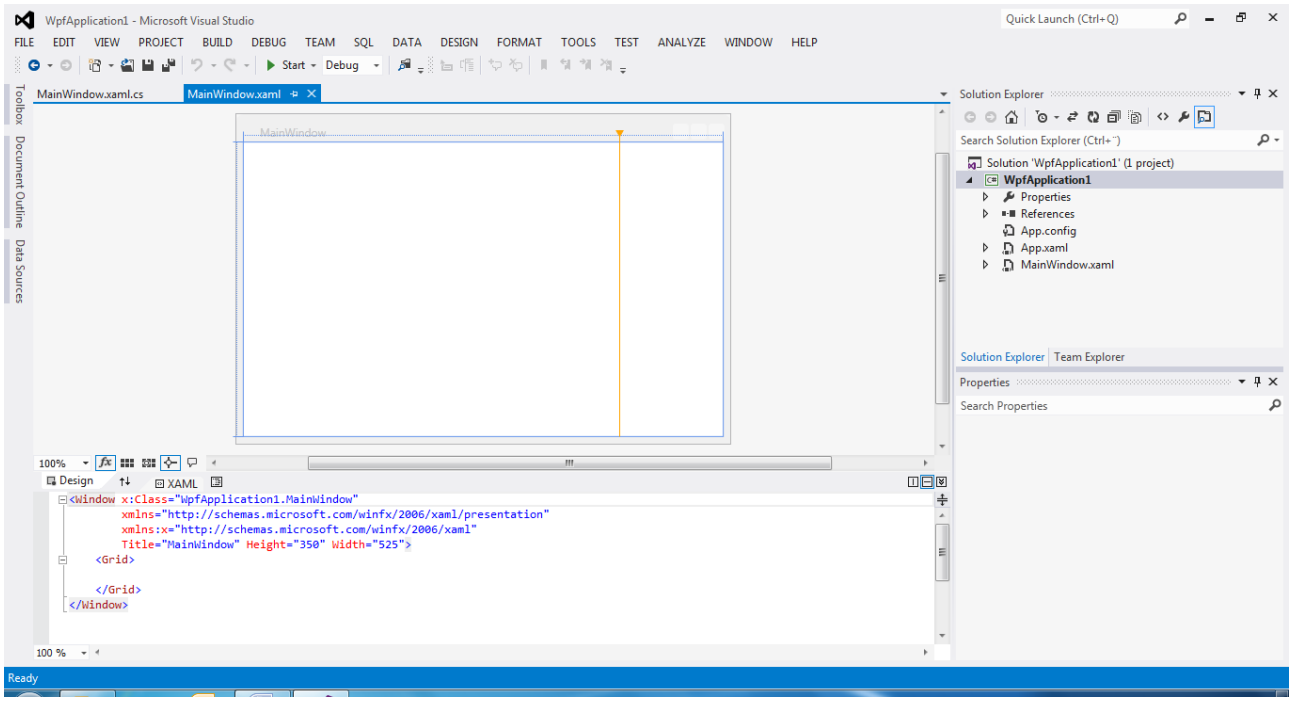


Another simple example (WPF)

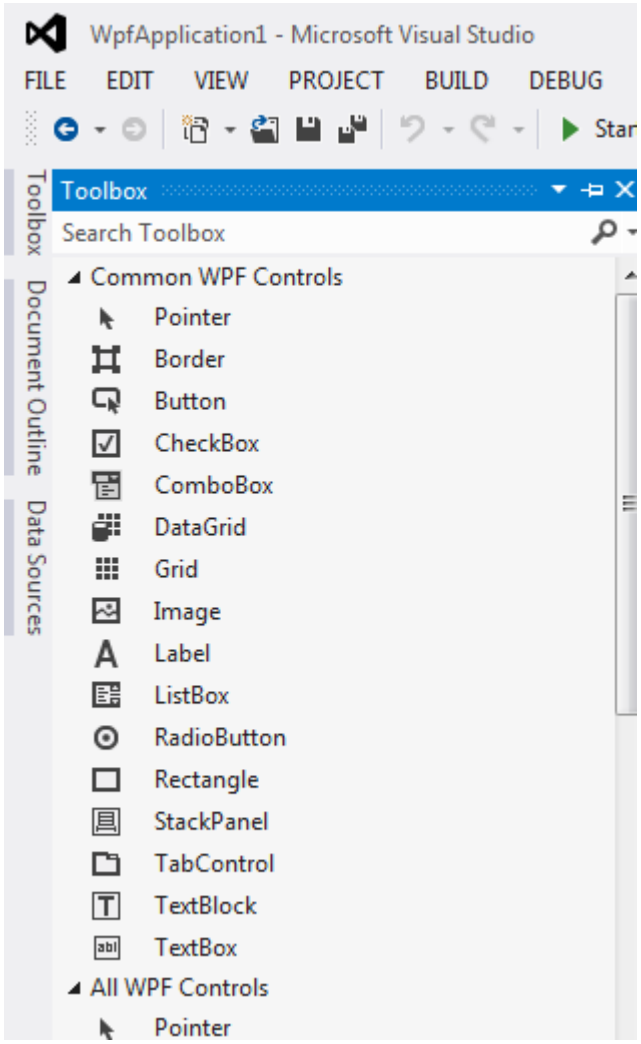
- Create a C# (WPF) project



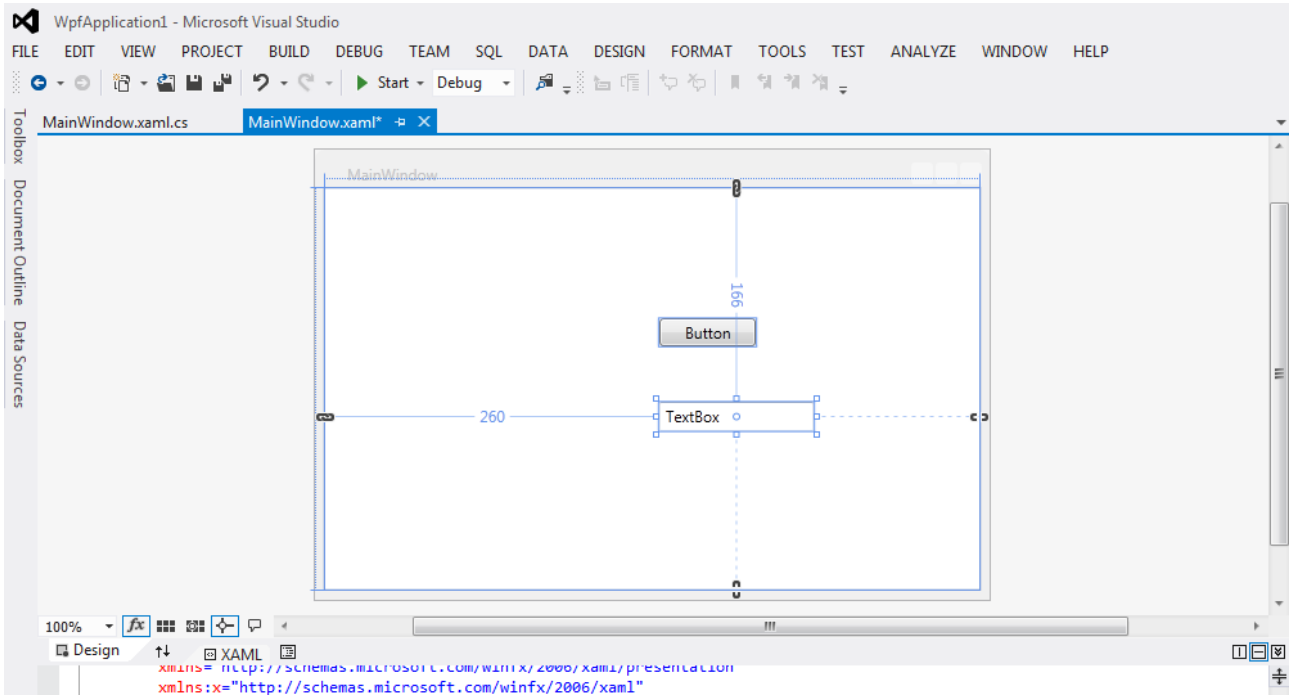
- Visual form (MainWindow.xaml) is created



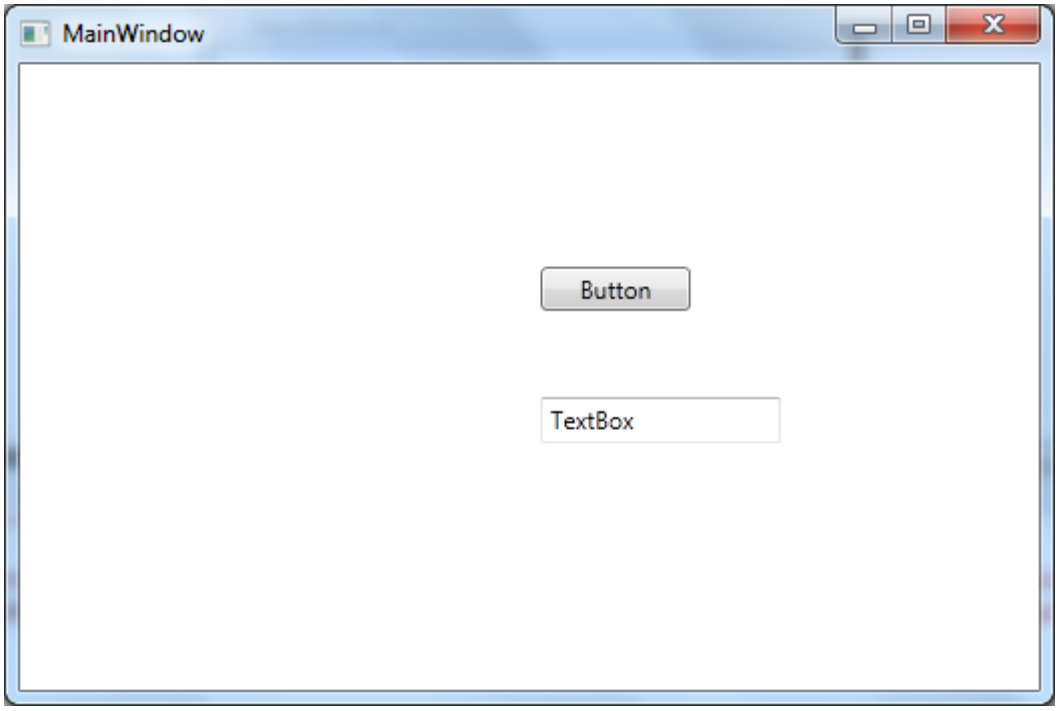
- Notice: Toolbox


WpfApplication1 - Microsoft Visual Studio
FILE
EDIT
VIEW
PROJECT
BUILD
DEBUG
Star
Toolbox
Search Toolbox
Common WPF Controls
Pointer
Border
Button
CheckBox
ComboBox
DataGrid
Grid
Image
Label
ListBox
RadioButton
Rectangle
StackPanel
TabControl
TextBlock
TextBox
All WPF Controls
Pointer
Toolbox
Document Outline
Data Sources

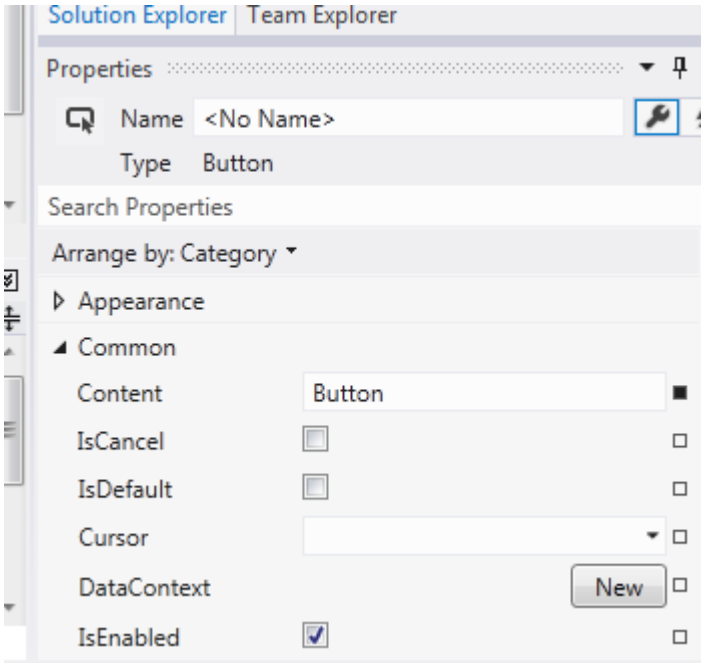
- Drag controls into the form



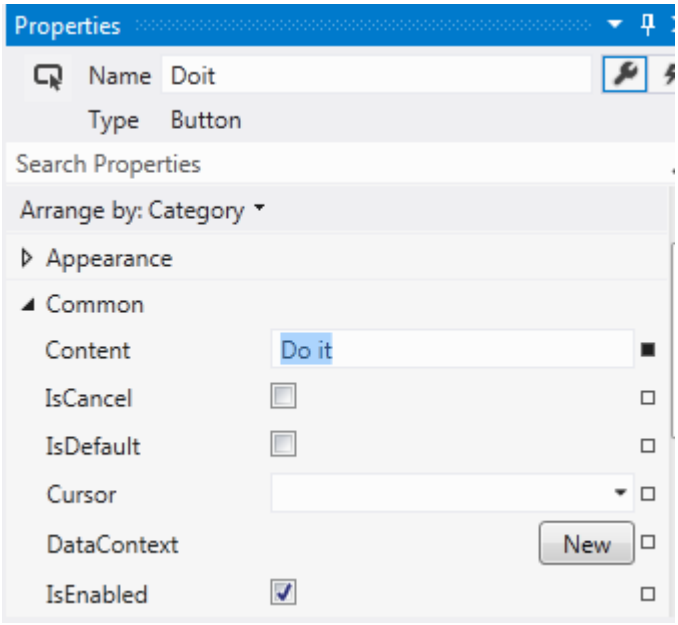
- Use Build | Build Solution (F6)
- Debug | Start Debugging (or F5) (or)
- Debug | Start Without Debugging (CTRL +F5)


MainWindow
Button
TextBox

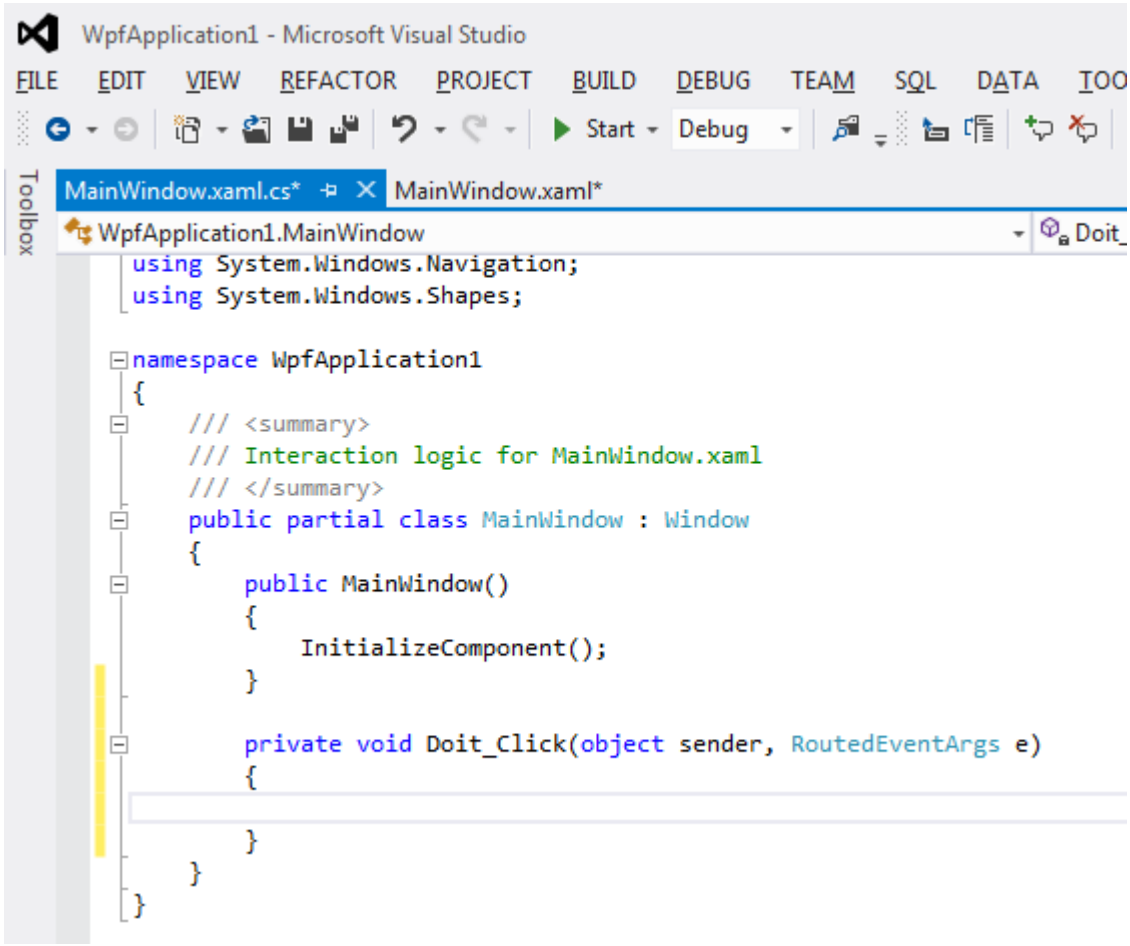
- Properties


Solution Explorer
Team Explorer
Properties
Name <No Name>
Type Button
Search Properties
Arrange by: Category
Appearance
Common
Content Button
IsCancel
IsDefault
Cursor
DataContext New
IsEnabled

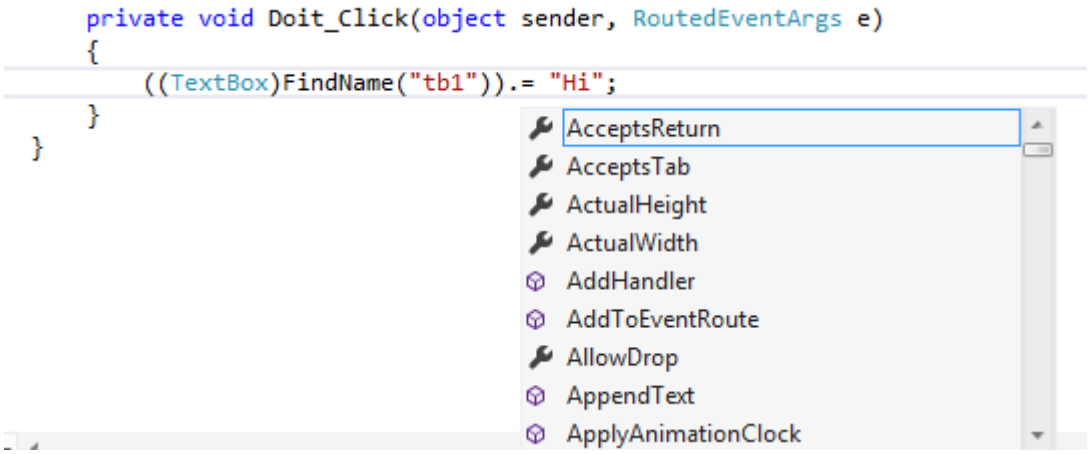
- Before double-click
    - Name your controls



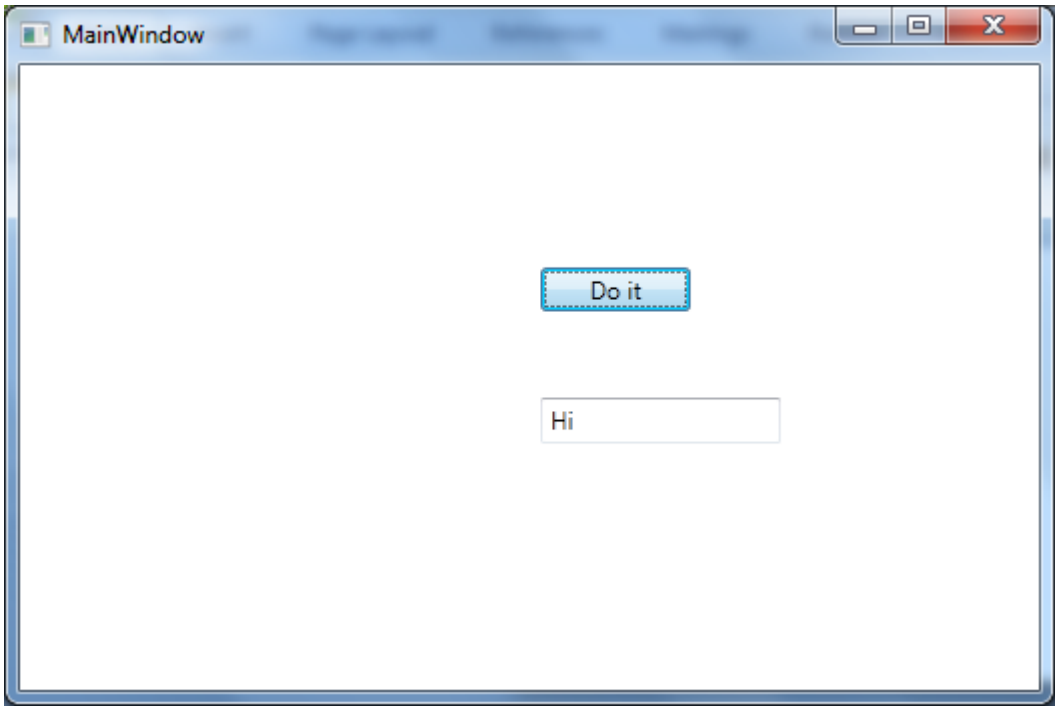
- Double-click in the visual form to add events



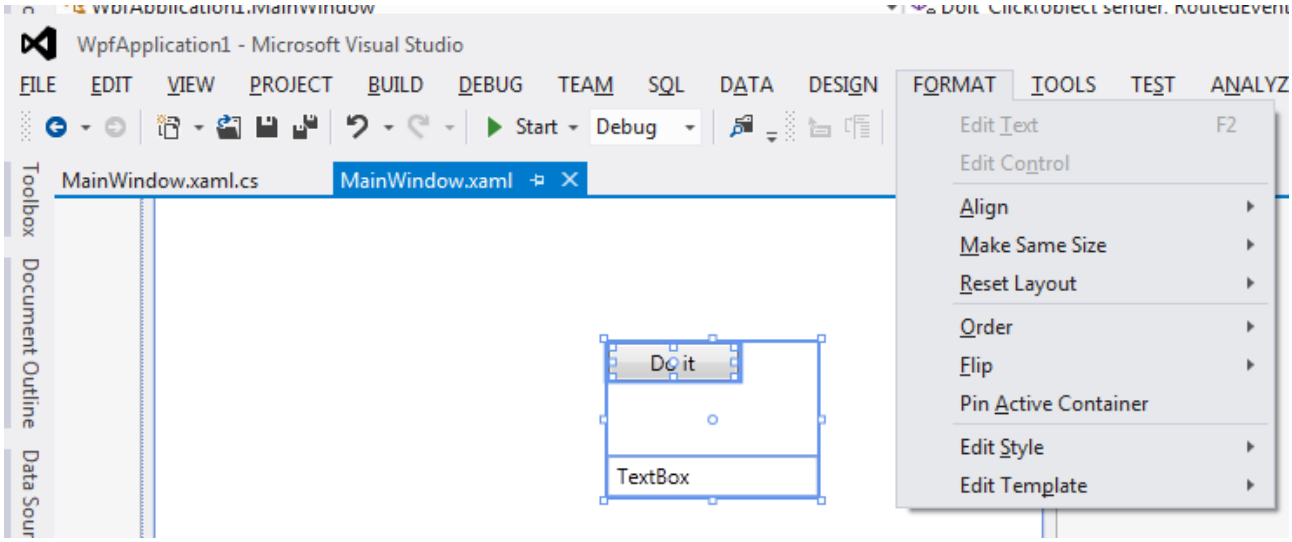
- Notice: Visual has (of course) intellisense



- Result – 1 minute work

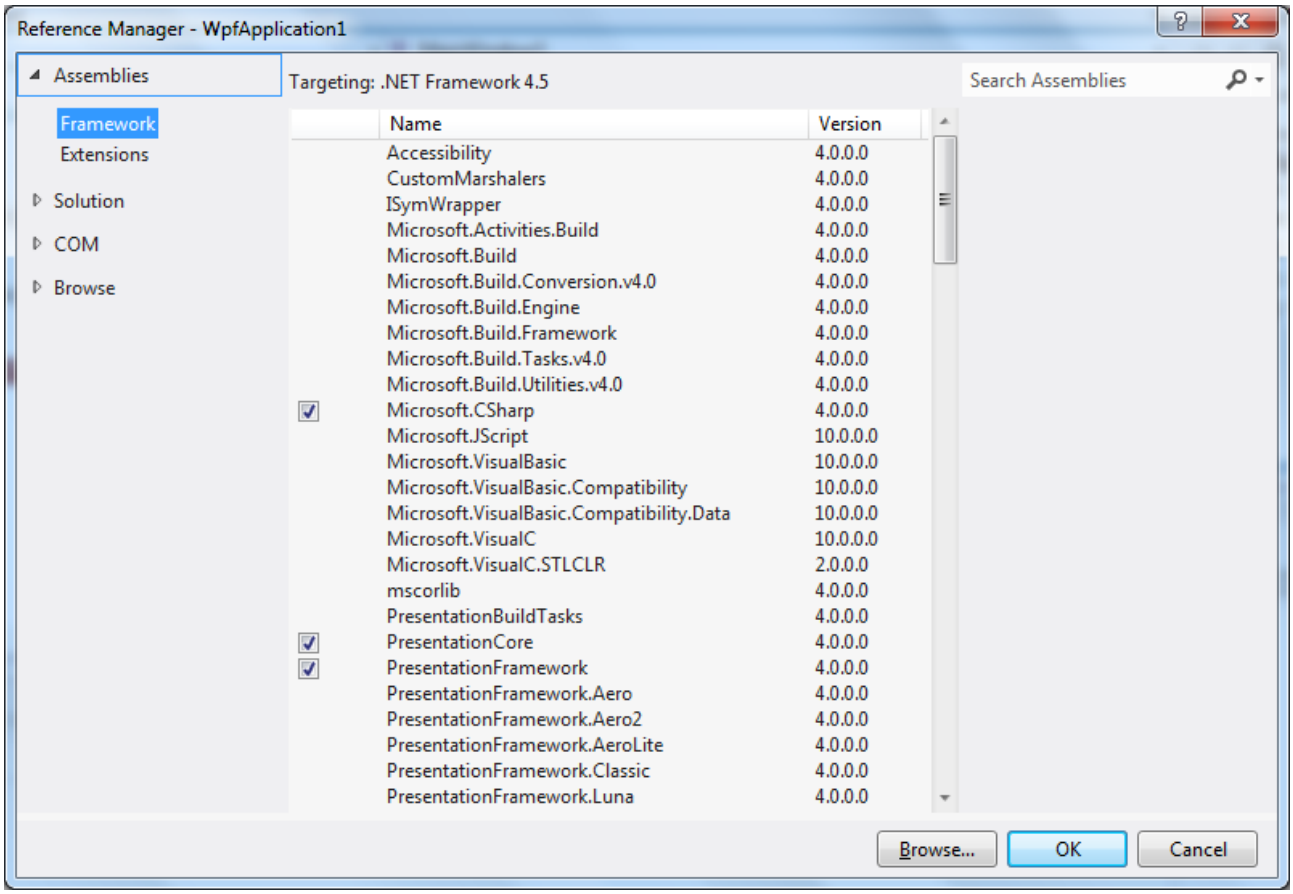


Layout

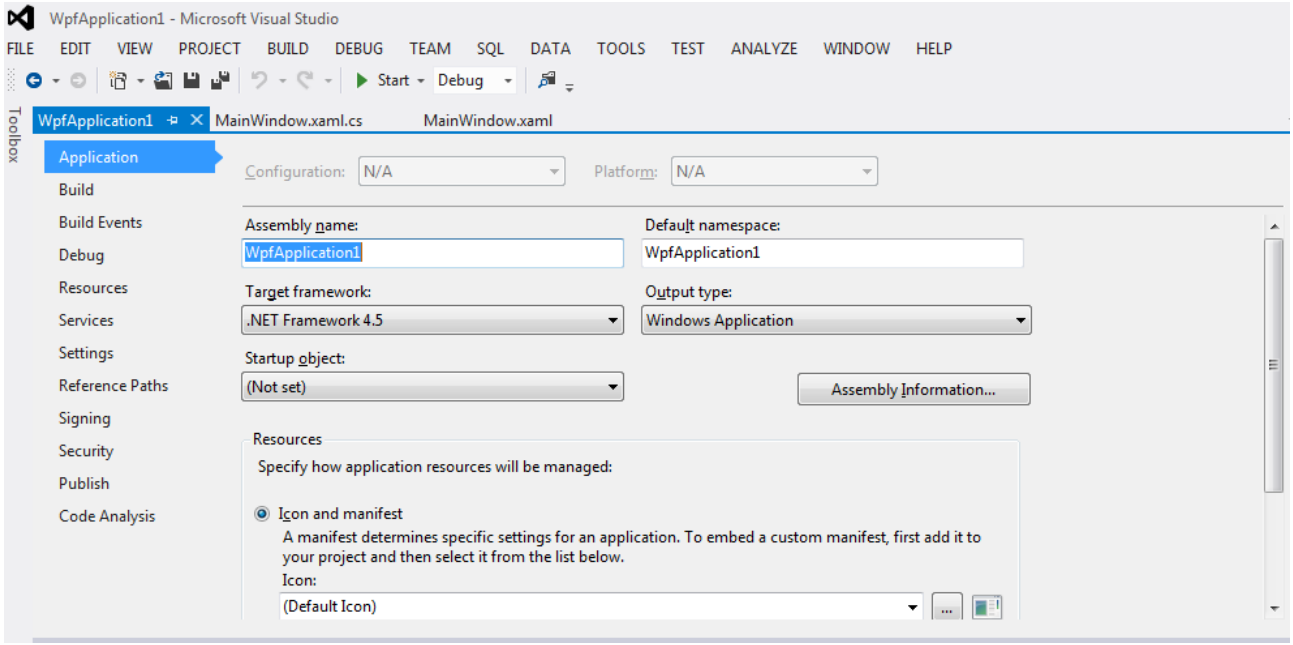


- Graphical layout
  - Use Format
  - For instance Align and ”Make same size”

Reference Manager



Project Properties



Refactor

- Notice: Active code file must be open

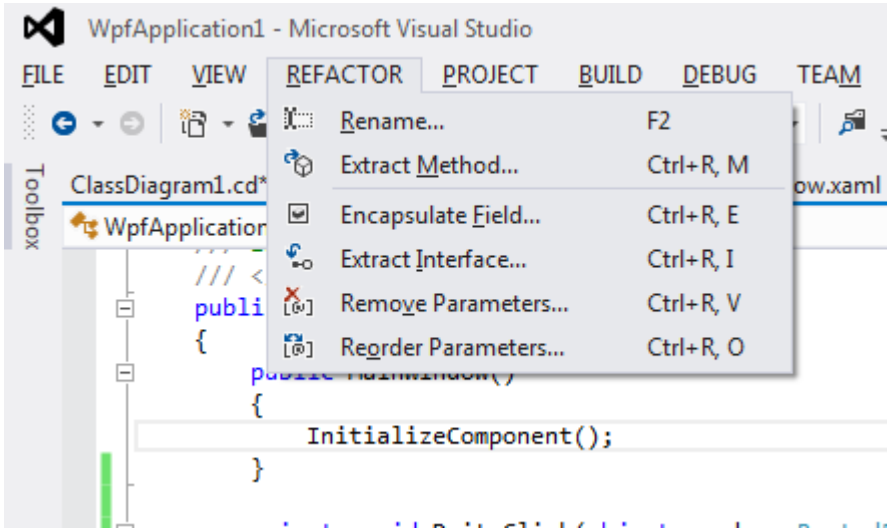


Table 2-2. *Visual Studio Refactorings*

Refactoring Technique	Meaning in Life
Extract Method	Allows you to define a new method based on a selection of code statements
Encapsulate Field	Turns a public field into a private field encapsulated by a C# property
Extract Interface	Defines a new interface type based on a set of existing type members
Reorder Parameters	Provide a way to reorder member arguments
Refactoring Technique	**Meaning in Life**
Remove Parameters	Remove a given argument from the current list of parameters (as you would expect)
Rename	Allows you to rename a code token (method name, field, local variable, and so on) throughout a project