A Brief Introduction to Swift

Table of Contents

C# Cheat Sheet by tschinz

This is a attempt to create a small easy-to-understand as well as easy-to-use cheat sheet for C#. C# is the programming language of Microsoft used in all of their devices. It is multiplatform and with help of Xamarin it can be deployed on a variety of devices. C# is a general-purpose, type-safe, object-oriented programming language. The goal of the language is programmer productivity. To this end, the language balances simplicity, expressiveness, and performance. The C# language is platform-neutral, but it was written to work well with the Microsoft .NET Framework. C# 6.0 targets .NET Framework 4.6.

License

Changelog

`v0.0.1` - TBD

A brief summary of the programming language of Microsotft called C# or .Net.

There is no guarantee that this reference is complete or correct. If you want to contribute or reporting an issue:

- Repo
- Issue

Contributor and References

- C# Guide
- Official Microsoft Reference
- Tutorial ADP.Net
- UML Class Diagram
- Inversion of Control
- Design Pattern

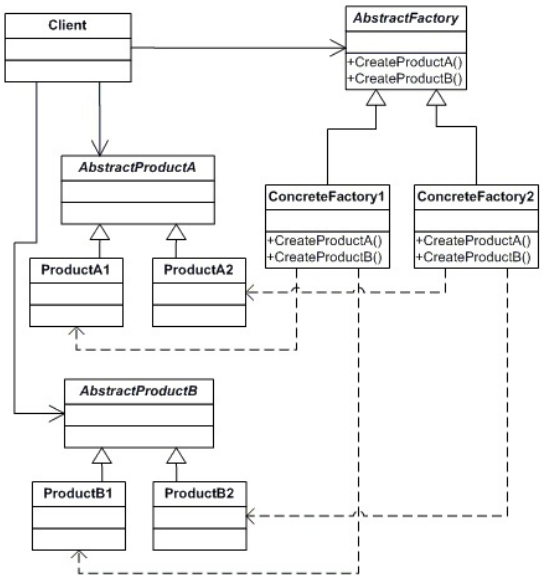
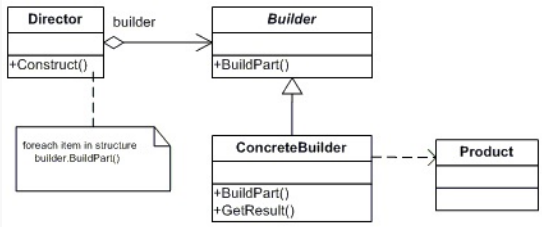
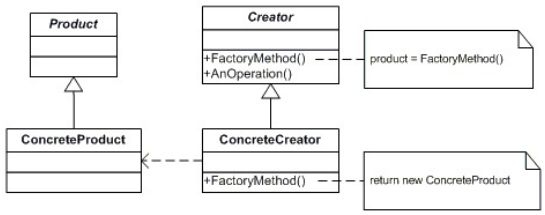
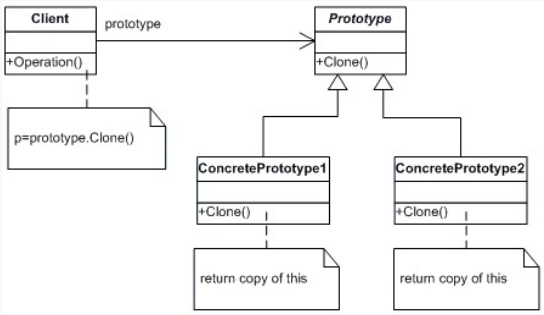
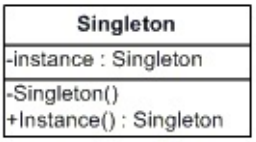
Thanks

Thanks goes to:

- Microsoft
- All contributors

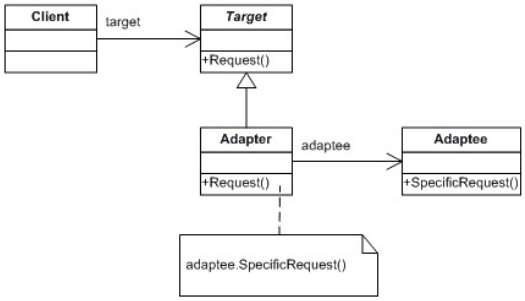
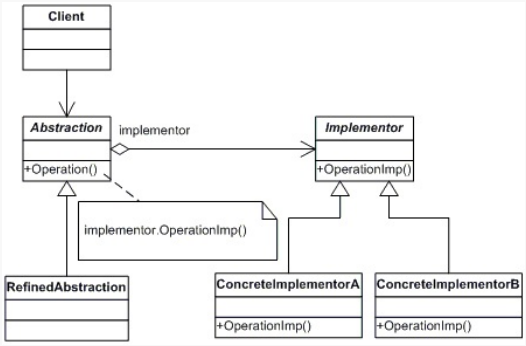
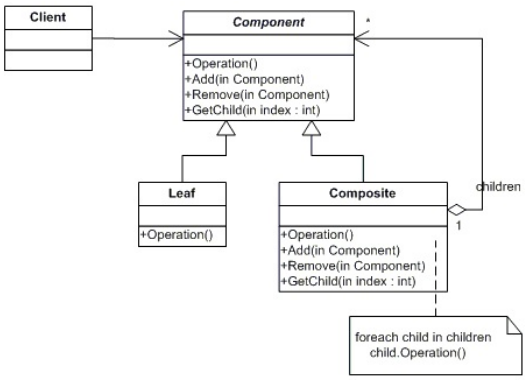
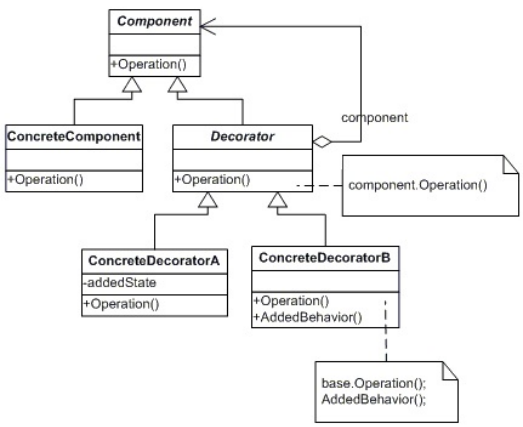
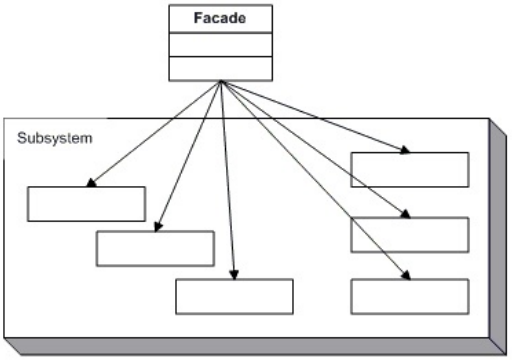
Design Pattern

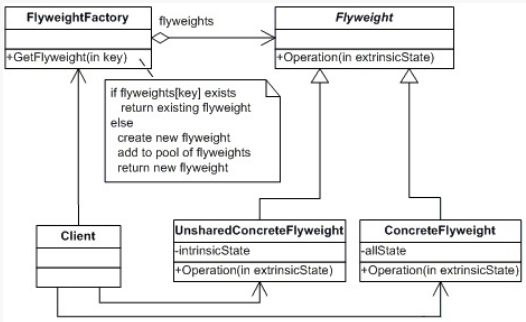
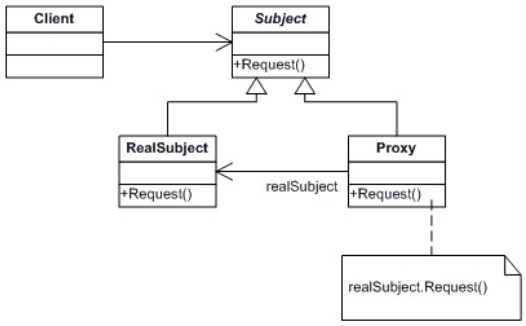
Creational Patterns

Pattern	Description	Image
Abstract Factory	Creates an instance of several families of classes	
Builder	Separates object construction from its representation	
Factory Method	Creates an instance of several derived classes	
Prototype	A fully initialized instance to be copied or cloned	
Singleton	A class of which only a single instance can exist	

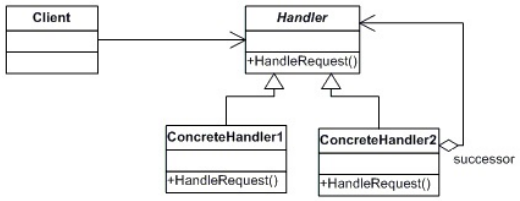
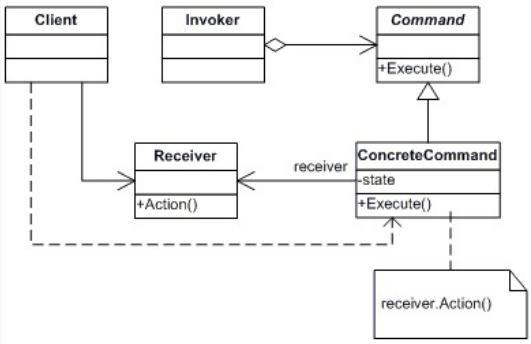
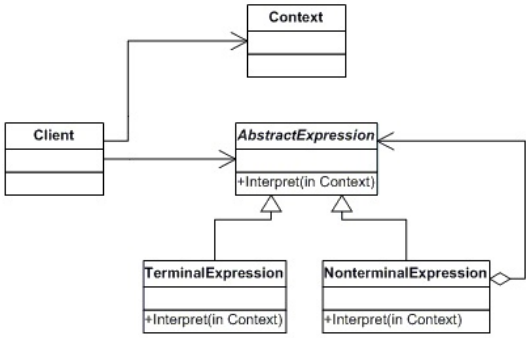
Structural Patterns

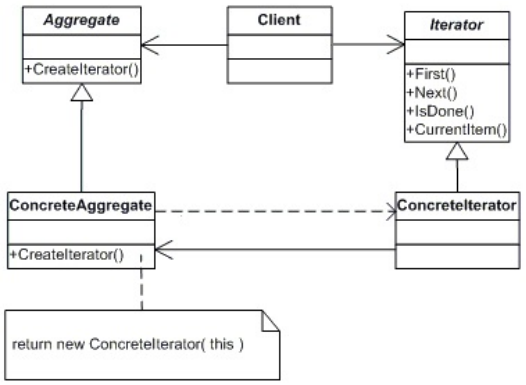
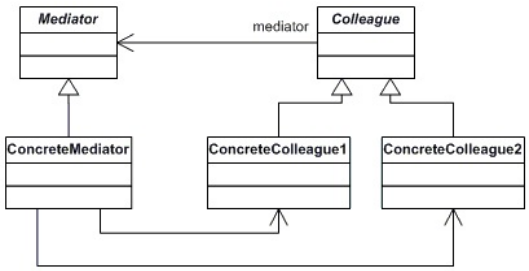
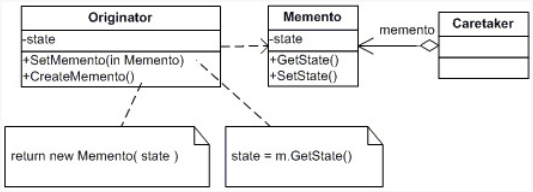
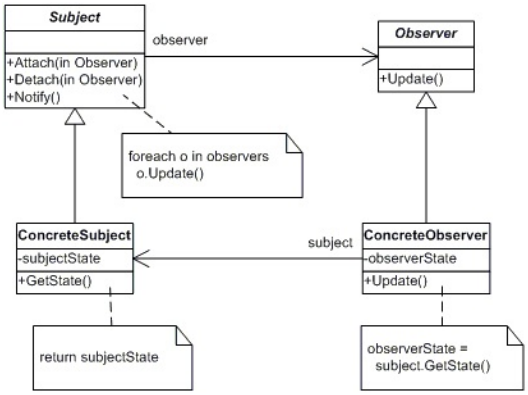
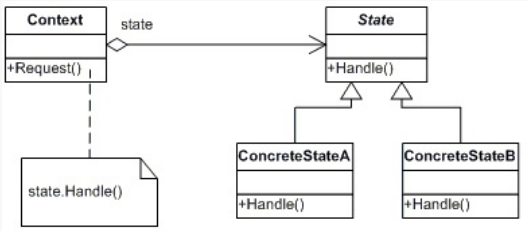
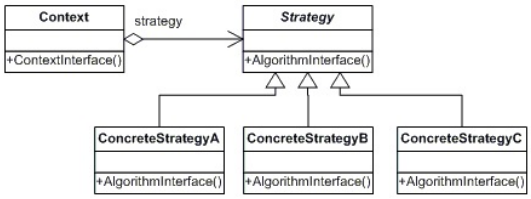
Pattern	Description	Image

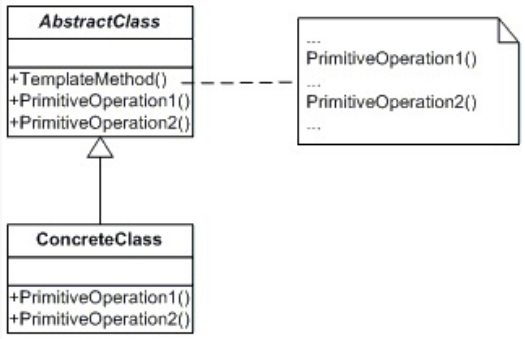
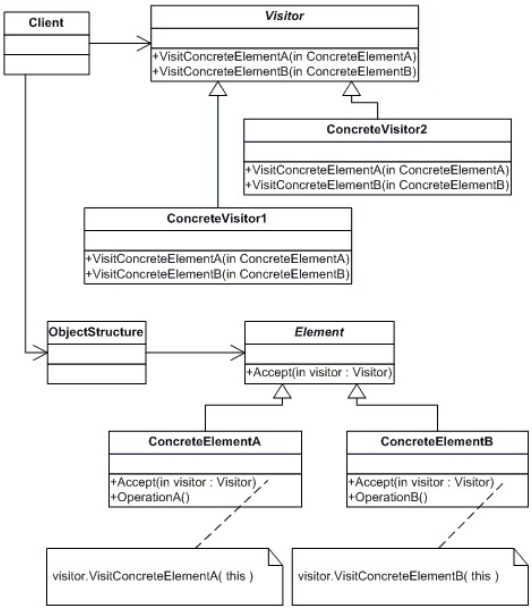
Adapter	Match interfaces of different classes	
Bridge	Separates an object's interface from its implementation	
Composite	A tree structure of simple and composite objects	
Decorator	Add responsibilities to objects dynamically	
Facade	A single class that represents an entire subsystem	

Flyweight	A fine-grained instance used for efficient sharing	
Proxy	An object representing another object	

Behavioral Patterns

Pattern	Description	Image
Chain of Resp.	A way of passing a request between a chain of objects	
Command	Encapsulate a command request as an object	
Interpreter	A way to include language elements in a program	

Iterator	Sequentially access the elements of a collection	
Mediator	Defines simplified communication between classes	
Memento	Capture and restore an object's internal state	
Observer	A way of notifying change to a number of classes	
State	Alter an object's behavior when its state changes	
Strategy	Encapsulates an algorithm inside a class	

Template Method	Defer the exact steps of an algorithm to a subclass	
Visitor	Defines a new operation to a class without change	

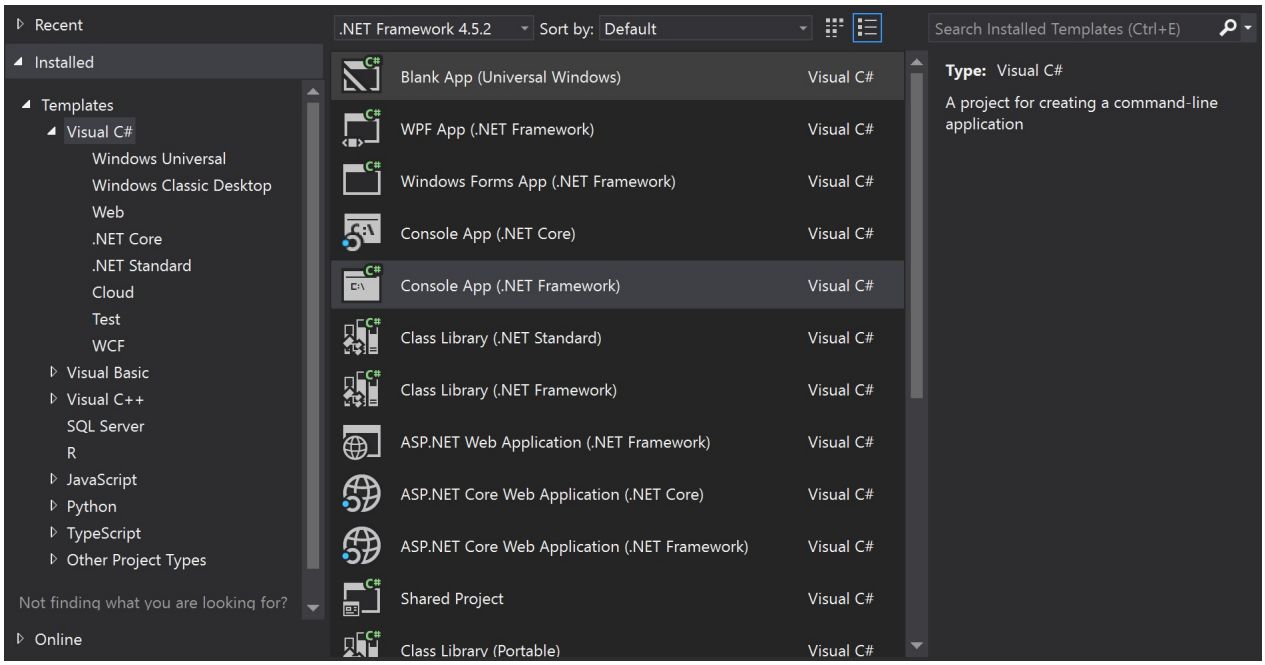
Visual Studio

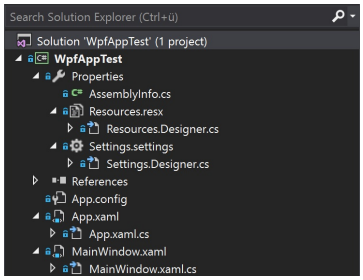
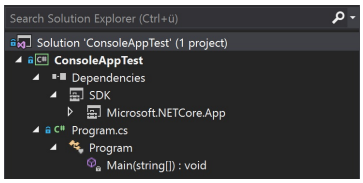
The Visual Studio integrated development environment (IDE) is a collection of development tools exposed through a common user interface. Some of the tools are shared with other Visual Studio languages, and some, such as the C# compiler, are unique to Visual C#.

Solutions

In VisualStudio only solutions can be used. Each project (aka solution) need to have all necessary file to be build and openend in VisualStudio.

A new Project / Solution can be created at `File` => `New` => `Project`



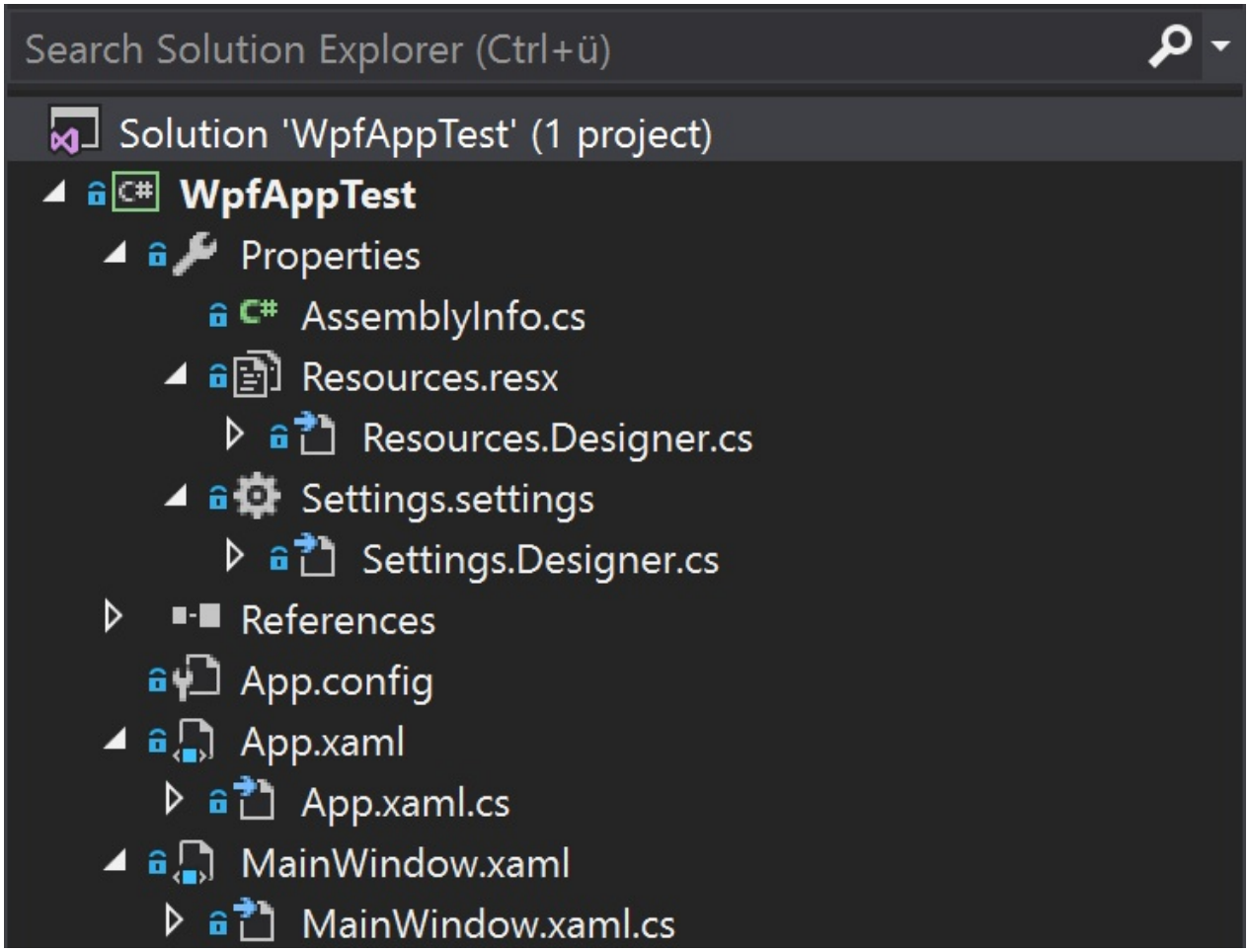
Project Type	Description	Solution image
Blank App (Universal Windows)	A project for a single-page Universal Windows Platform (UWP) app that has no predefined controls or layout. Best if you know what you're doing.	
WPF App (.NET Framework)	Classic graphical desktop application based on the .NET Framework. Also called Windows Presentation Foundation client application.	
Console App (.NET Core)	New Console app based on the .NET Core. A project for creating a command-line application that can run on Windows, Linux, MacOS. (Doesn't creates an exe file).	

Console App (.NET Framework) | A project for creating a command-line application. This will only work on Windows and will create an exe directly.

- One **Solution** can contain many **Projects**

WPF App

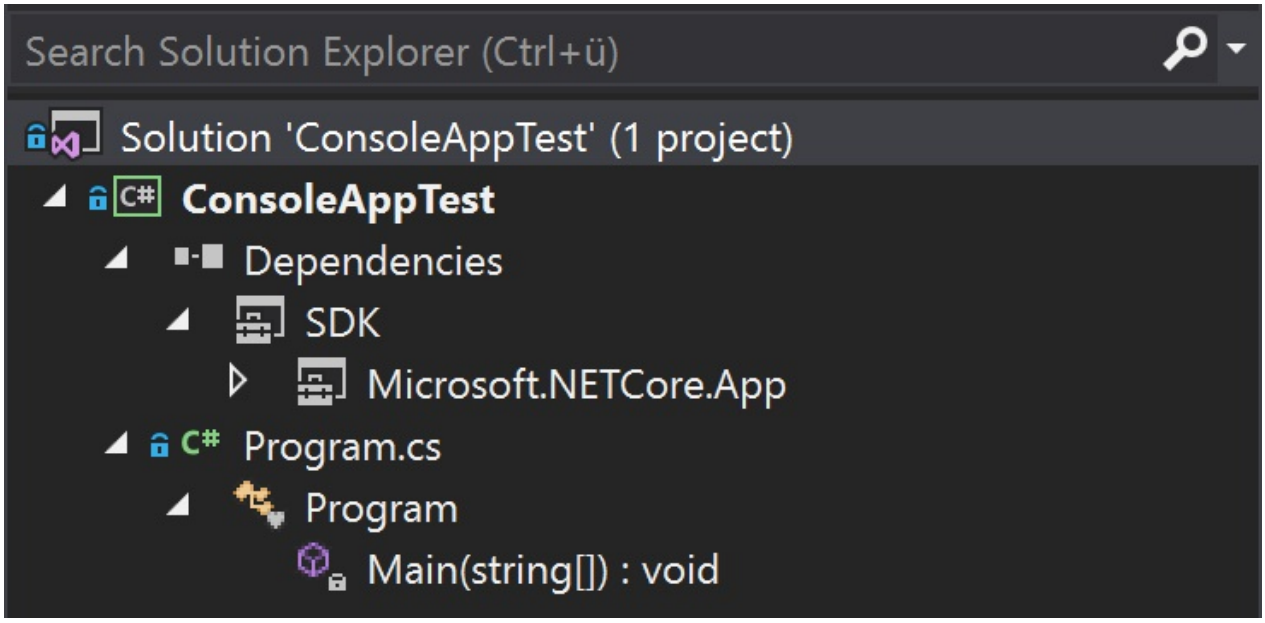
A WPF App already has a predefined structure


Search Solution Explorer (Ctrl+ü)
Solution 'WpfAppTest' (1 project)
WpfAppTest
Properties
AssemblyInfo.cs
Resources.resx
Resources.Designer.cs
Settings.settings
Settings.Designer.cs
References
App.config
App.xaml
App.xaml.cs
MainWindow.xaml
MainWindow.xaml.cs

Type	Element	Description
Properties	AssemblyInfo.cs	AssemblyInfo.cs contains information about your assembly, like name, description, version, etc. If you delete it, your assembly will be compiled with no information.
Properties	Resources	This is a database like structure for storing resources needed in the program. This can be Files, Images, String Values, Text Files.
Properties	Settings	Application settings allow you to store and retrieve property settings and other information for your application dynamically.
References		Contains all necessary namespaces and third party libraries used in the Application. It is better to use NuGet packages found in the Project settings.
`App.config`		is an XML file with many predefined configuration sections available and support for custom configuration sections. A "configuration section" is a snippet of XML with a schema meant to store some type of information.
`App.xaml`		`App.xaml` also contains a `.cs` file. This is the main entry point launching the gui.
`MainWindow.xaml`		`MainWindow.xaml` also contains a `.cs` file. Each of these files represent a part of the GUI.

In the Project Settings all above elements can be set in one single location

Console App (.NET Core)



Type	Description
Dependencies	List of all needed dependencies and third party libraries for the program
`Program.cs`	Main program C# file, one needs to contain the Main method as entry point of the program.

Programming Basics

Comments

```
// This is a comment
/* This is a multiline comment
   They can not be nbested! */
```

Comment Links

```
// MARK: Test appears in Xcode selector
// TODO: Write your todo here
// FIXME: Write your bug here (or fix is directly)
```

Importing namespaces

```
using System; // import namespace System
Console.WriteLine(x); // use function of namespace System

System.Console.WriteLine(x); // using without importin namespace
```

Types

For a complete guide to 64-bit changes, please see the transition document.

Default Swift Types

Swift Type	Values
Int Int32, Int64, UInt8, UInt16	`100` Dezimal `1_000_000` Dezimal `0b1001` Binary `0o85` Octal `0xFFE3` Hexadezimal

Double	`3.14159265`
	`3.141_592_65`
	`1.25e2 = 125.0`
	`1.25e-2 == 0.0125`

Bool	`true` `false`
String	`"This is a String"` `"This is a String including a \(varname)"`

C-Types vs Swift Types

C Type	Swift Type
bool	CBool
char, signed char	CChar
unsigned char	CUnsignedChar
short	CShort
unsigned short	CUnsignedShort
int	CInt
unsigned int	CUnsignedInt
long	CLong
unsigned long	CUnsignedLong
long long	CLongLong
unsigned long long	CUnsignedLongLong
wchar_t	CWideChar
char16_t	CChar16
vchar32_t	CChar32
float	CFloat
double	CDouble

From the docs

Operators

Swift supports most standard C operators and improves several capabilities to eliminate common coding errors.

Arithmetic operators (`+` , `-` , `*` , `/` , `%` and so forth) detect and disallow value overflow, to avoid unexpected results when working with numbers that become larger or smaller than the allowed value range of the type that stores them.

Arithmetic Operators

Operator	Purpose
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder also works on float `8 % 2.5 // equals 0.5`

Comparative Operators

Operator	Purpose
==	Equal to
===	Identical to
!=	Not equal to
!==	Not identical to
~=	Pattern match
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Assignment Operators

Operator	Purpose
=	Assign
+=	Addition
-=	Subtraction
*=	Multiplication
/=	Division
%=	Remainder
&=	Bitwise AND
|=	Bitwise Inclusive OR
^=	Exclusive OR
<<=	Shift Left
>>=	Shift Right
&&=	Logical AND
||=	Logical OR

Increment and Decrement Operators

Operator	Purpose
++	Addition
--	Subtraction

```
++x //increments variable **before** returning it's value
x-- //increments variable **after** returning it's value
```

Logical Operators

Operator	Purpose
!	NOT
&&	Logical AND
||	Logical OR

Range Operators

Operator	Purpose
..<	Half-open range
...	Closed range

```
for index in 1..<3 {} // 1 to 3 excluding 3
for index in 1...3 {} // 1 to 3 inluding 3
```

Bitwise Operators

Operator	Purpose
&	Bitwise AND
|	Bitwise Inclusive OR
^	Exclusive OR
~	Unary complement (bit inversion)
<<	Shift Left
>>	Shift Right

Overflow and Underflow Operators

Typically, assigning or increment an integer, float, or double past it's range would result in a run-time error. However, if you'd instead prefer to safely truncate the number of available bits, you can opt-in to have the variable overflow or underflow using the following operators:

Operator	Purpose
&+	Addition
&-	Subtraction
&*	Multiplication
&/	Division
&%	Remainder

Example for unsigned integers (works similarly for signed):

```
var willOverflow = UInt8.max        // willOverflow = 255
willOverflow = willOverflow &+ 1    // willOverflow = 0

var willUnderflow = UInt8.min       // willUnderflow = 0
willUnderflow = willUnderflow &- 1 // willUnderflow = 255
```

Another example to show how you can prevent dividing by zero from resulting in infinity:

```
let x = 1
let y = x &/ 0 // Division by zero y = 0
```

Other Operators

Operator	Purpose
??	Nil coalescing (take left if not nil else right value)
?:	Ternary conditional
!	Force unwrap object value
?	Safely unwrap object value