Instalación de C#

Índice

Instalación de C#

1 Introducción

El lenguaje C#, en el momento de escribir este documento, es soportado por varios *frameworks* o entornos de programación. El primero es *.NET Framework*, que permanecerá en un principio como el entorno dedicado solo para Windows, si bien será declarado obsoleto a medio plazo. *Mono*, que principalmente se emplea para desarrollo multiplataforma para dispositivos móviles. Finalmente, *.NET Core* (en el futuro, solo *.NET*), será el *framework* multiplataforma que terminará por sustituir a todos los anteriores. Es por esto que utilizaremos este último, ya que es el único con vocación de completitud de funcionalidad e implementación para todos los dispositivos.

2 Descarga del SDK

Para descargar el paquete, será necesario acceder a la URL de *dotnet*[1]. Si elegimos *download* (descargar) para el SDK (*Software Development Kit*), por defecto nos descargará el instalador para nuestra plataforma, bien sea esta Windows, Linux o Mac. En este documento no emplearemos los instaladores, pues a) no descargan los binarios en una carpeta verdaderamente accesible por el usuario, lo que hace que b) sea compleja su localización para la configuración de cualquier otro *software* que dependa de él.

Si seleccionamos *all downloads* (todas las descargas), podremos comprobar que para todas las plataformas están disponibles tanto instaladores como binarios comprimidos, siendo estos los que escogeremos.

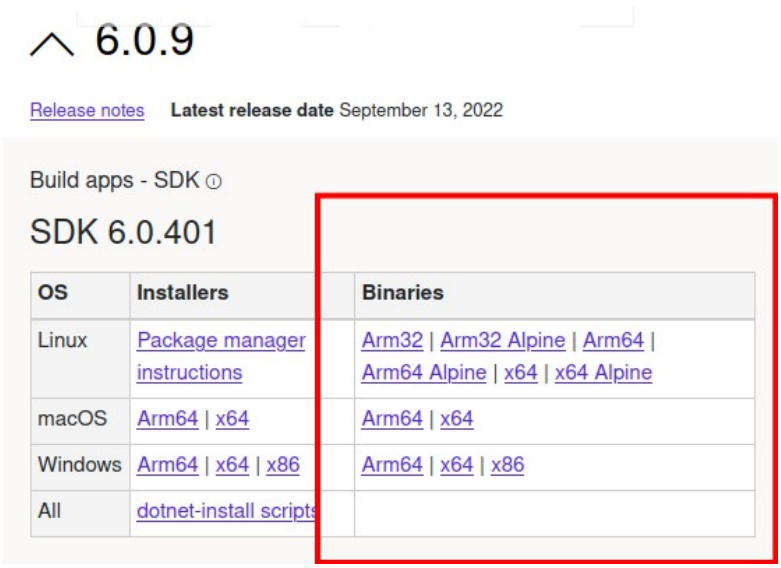


Figura 1: A la derecha se encuentran los binarios comprimidos.

De todos los posibles paquetes, escogeremos el que se corresponda con la CPU de nuestro dispositivo, típicamente x64 (Intel x86 de 64bits).

1 URL para descargar .NET: http://dotnet.microsoft.com/

3 Instalación

Una vez descargado, procederemos a la instalación. En la carpeta de usuario, procederemos a crear una subcarpeta llamada *bin/*. Dentro de esta carpeta descomprimiremos el paquete descargado. Por ejemplo, siguen las instrucciones para *Linux*, que serán muy similares a las de *Mac*, siendo las más diferentes las de *Windows*[2].

```
$  cd
$  pwd
/home/baltasarq
$ mkdir bin
$ cd bin
$ cp ~/Downloads/dotnet-sdk-6.0.401-linux-x64.tar.gz .
$ tar -xvzf dotnet-sdk-6.0.401-linux-x64.tar.gz
```

Las mismas instrucciones para *Windows*.

```
c:\users\baltasarq> cd \users\baltasarq
c:\users\baltasarq> mkdir bin
c:\users\baltasarq> cd bin
c:\users\baltasarq> copy ~/Downloads/dotnet-sdk-6.0.401-win-x64.zip
c:\users\baltasarq> unzip ~/Downloads/dotnet-sdk-6.0.401-win-x64.zip
```

Por supuesto, de lo que se trata básicamente es de crear el directorio *bin/* en el directorio *home* del usuario, y esto en cualquiera sistema operativo de los comentados se puede hacer mediante las herramientas visuales. La única posible dificultad en *Windows* es abrir la carpeta de usuario: esto se puede hacer fácilmente pulsando en la primera flecha de la barra de *path* o de dirección de directorios.

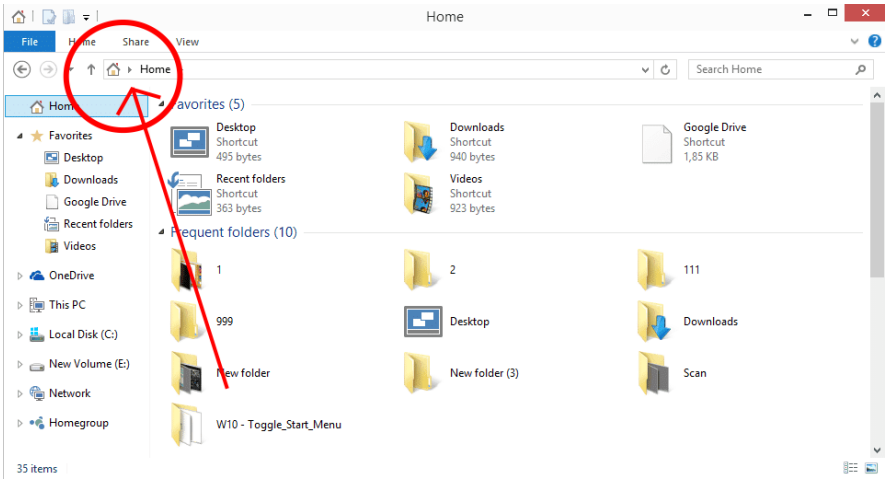


Figura 2: Acceso a la carpeta de home del usuario.

Una vez con la carpeta descomprimida, la renombramos a tan solo "dotnet", y para poder manejarnos directamente con el programa, modificamos la variable de usuario PATH. En Linux lo conseguimos añadiendo la línea "SET PATH=$PATH:~/home/<usuario>/bin/dotnet" al final del archivo *~/.bashrc* (si se utiliza el shell por defecto, *bash*) o *~/.zshrc* si se utiliza *zsh*. Por ejemplo, si

2 Pueden descargarse los ciertos ejecutables como unzip.exe en: http://stahlworks.com/dev/index.php?tool=zipunzip

el nombre del usuario es “baltasar” y el shell es *bash*:

```
$ echo “SET PATH=$PATH:~/home/baltasarq/bin/dotnet” >> ~/.bashrc
$ source ~/.bashrc
```

En Windows, es posible acceder a la información del sistema y desde allí cambiar las variables de entorno del usuario (no las del sistema), añadiendo “c:\users\<usuario>\bin\dotnet” a la variable *path*, aunque también puede hacerse desde la línea de comando. Por ejemplo, si el usuario es *baltasarq*:

```
c:\> setx path %path%;”c:\users\baltasarq\bin\dotnet”
```

4 Programación en C# con .NET Core

Una vez instalado, ya es posible crear y compilar un programa con C#. La herramienta *dotnet* centraliza todos los procesos: creación del proyecto, compilación y ejecución. Concretamente, utilizaremos los comandos *dotnet new*, *dotnet build*, y *dotnet run*. Veremos este proceso a continuación.

4.1 Creando el proyecto

El comando necesario es *dotnet new*, sin embargo este comando tiene diferentes opciones relacionados con las plantillas para nuevos proyectos que soporta. Probablemente, la plantilla más utilizada sea *console*, que como su nombre sugiere permite crear una aplicación de consola. Para poder acceder a las plantillas disponibles, utilizaremos la opción *--list*, y para poder instalar plantillas nuevas *–install*.

La opción *list* admite una palabra para realizar búsquedas en las plantillas instaladas (si no se indica las lista todas), y esta palabra puede ser tan completa como se desee (“cons” y “console” provocan el mismo resultado). La opción *update-apply* comprueba que todas las plantillas estén actualizadas, y si no es así, las actualiza. La opción *search* permite buscar en el repositorio de plantillas.

```
$ dotnew new –list cons
These templates matched your input: 'console'

Template Name  Short Name  Language    Tags
-------------  ----------  ----------  --------------
Console App    console     [C#],F#,VB  Common/Console

$ dotnet new --update-apply
All template packages are up-to-date.

$ dotnet new --search json
Searching for the templates...
Matches from template source: NuGet.org
These templates matched your input: 'json'

Template Name                       Short Name       Language
-------------                       ----------       --------
...
SpecFlow Configuration JSON File    specflow-json    [C#]
SpecFlow Configuration JSON File    specflow-json    [C#]
```

```
templatejson                          templatejson    [C#]


To use the template, run the following command to install the package:
   dotnet new --install <PACKAGE_ID>
Example:
   dotnet new --install SpecFlow.Templates.DotNet
```

Finalmente, para crear un proyecto utilizamos *new* con el nombre corto de la plantilla y *-o* o *--output* para indicar el nombre del mismo. En el siguiente código creamos un proyecto nuevo llamado "holamundo":

```
$ dotnet new console -o holamundo
The template "Console App" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on /home/baltasarq/tmp/holamundo/holamundo.csproj...
  Determining projects to restore...
  Restored /home/baltasarq/tmp/holamundo/holamundo.csproj (in 161 ms).
Restore succeeded.

$ cd holamundo
$ ls
holamundo.csproj  obj  Program.cs
$ cat Program.cs
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```

4.2 Construyendo la aplicación

Si nos encontramos dentro del directorio de la aplicación, solo es necesario *dotnet build* para construir la aplicación.

```
$ pwd
/home/baltasarq/tmp/holamundo
$ dotnet build
Microsoft (R) Build Engine version 17.0.0+c9eb9dd64 for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

  Determining projects to restore...
  All projects are up-to-date for restore.
  holamundo -> /home/baltasarq/tmp/holamundo/bin/Debug/net6.0/holamundo.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:03.38

$ ls
bin  holamundo.csproj  obj  Program.cs
$ ls bin
Debug
$ ls bin/Debug
```

```
net6.0
$ ls bin/Debug/net6.0
holamundo  holamundo.deps.json  holamundo.dll  holamundo.pdb
$ bin/Debug/net6.0/holamundo
Hello, World!
```

Por defecto, la aplicación se compila en modo *Debug*, es decir, incorporando información de depuración. Para una aplicación final, debe indicarse la opción *--configuration* (o *-c*), con *Release*, que genera una aplicación lista para ser distribuida.

```
$ dotnet build --configuration Release
Microsoft (R) Build Engine version 17.0.0+c9eb9dd64 for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

  Determining projects to restore...
  All projects are up-to-date for restore.
  holamundo -> /home/baltasarq/tmp/holamundo/bin/Release/net6.0/holamundo.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:01.19
$ ls bin
Debug  Release
$ ls bin/Release/net6.0
holamundo  holamundo.deps.json  holamundo.dll  holamundo.pdb
$ bin/Release/net6.0/holamundo
Hello, World!
```

4.3 Ejecución

La ejecución de código, como hemos visto, se puede lograr "buscando" el ejecutable en el subdirectorio correcto. Sin embargo, es mucho más sencillo emplear la opción *run*:

```
$ dotnet run
Hello, World!
```

dotnet construirá el ejecutable si es necesario, y después lo ejecutará.

5 Instalación de un IDE

Existen muchos editores que pueden configurarse fácilmente para soportar programación con C#. En este documento trataremos *Geany*, *Visual Studio Code*, *Rider* y *Visual Studio*.

5.1 Geany

Geany es un editor de textos multiplataforma con cierto soporte para proyectos. Tiene soporte para C#, entre otros lenguajes de programación, y se puede configurar fácilmente para compilar y ejecutar desde el propio editor, o incluso desde el terminal integrado.

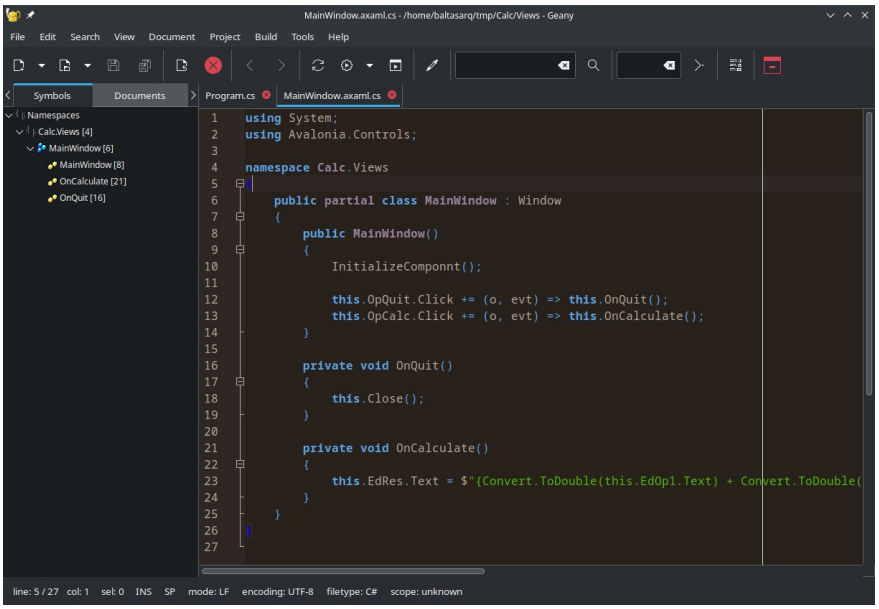
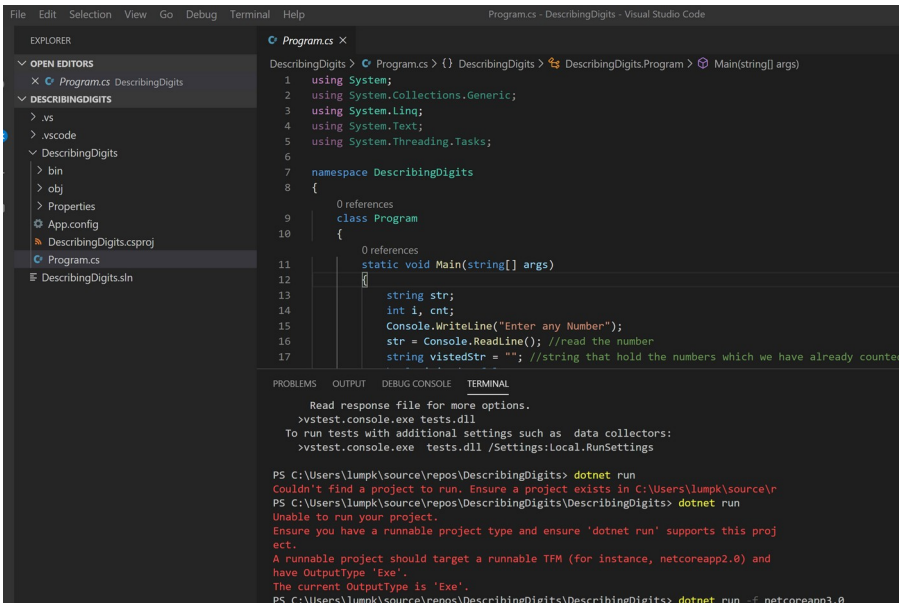


Figura 3: El editor Geany en acción.

En el menú de Tools (*herramientas*), podemos configurar las opciones compile y run para que ejecuten *dotnet build* y *dotnet run* respectivamente.

5.2 Visual Studio Code

Visual Studio Code es un editor multiplataforma con soporte para proyectos. El soporte para C# es muy completo, soportando no solo el habitual soporte de localización de las líneas de error desde el mensaje dado, sino que a través de la herramienta *OmniSharp*, es capaz de ofrecer incluso refactorización de código.



5.3 Visual Studio

Visual Studio de Microsoft es la opción de la compañía con dos versiones: la profesional (de pago), y la de la comunidad (gratuita), con el solo defecto de que solo funciona bajo Windows.

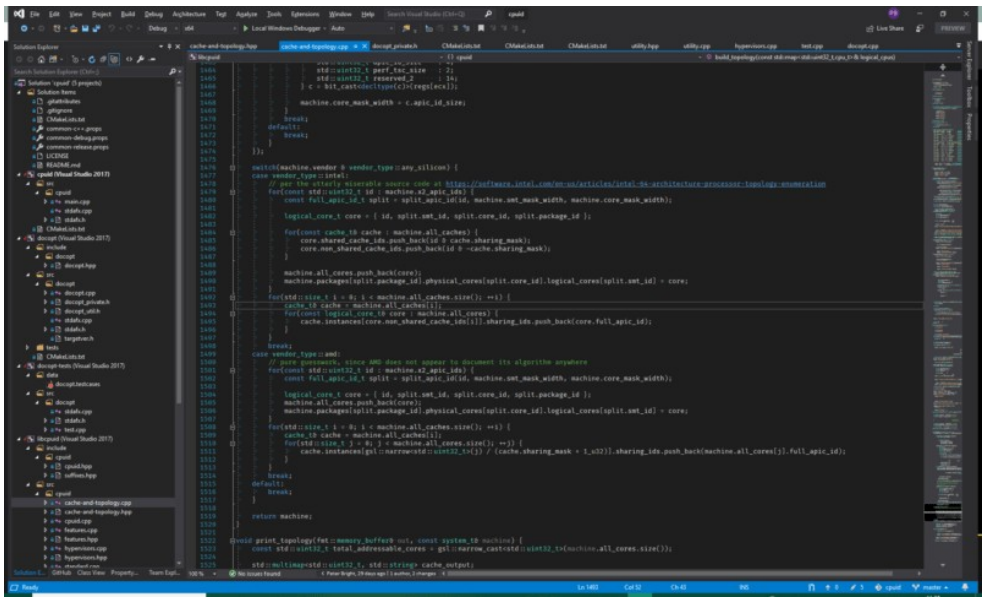


Figura 4: Visual Studio en acción.

5.4 Rider

IntelliJ Rider es una interesante oferta de la mano de IntelliJ para C#. Aunque no tiene una versión gratuita, proporciona la herramienta sin coste para alumnos y profesores. Además, las llamadas *early access* son gratuitas.

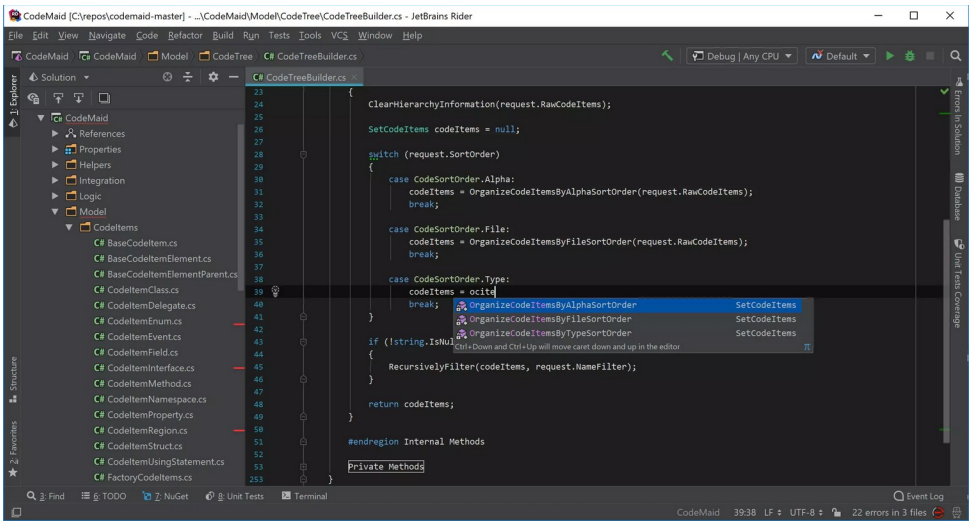


Figura 5: Rider en acción.

6 Referencias

1. .NET: http://dotnet.microsoft.com
2. Visual Studio: http://visualstudio.microsoft.com
3. Visual Studio Code: http://code.microsoft.com
4. Rider: https://www.jetbrains.com/rider/
5. Geany: https://www.geany.org/