

C# et .Net Core

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

`elmouelhi.achref@gmail.com`



Plan

- 1 Introduction
- 2 Avant de commencer
- 3 Premier projet Console .NET Core
- 4 Aspect multi-langages du .NET Core
- 5 Afficher un message dans la console
- 6 Commentaires
- 7 Console
- 8 Référence

- Langage de programmation
 - orienté objet,
 - fortement typé,
 - doté d'un typage statique, dynamique et générique,
 - Dérivé du **C++** et influencé par **Java** : syntaxe et logique assez proches.
- Présenté officiellement en 2002 par **Microsoft**
- Créé par **Anders Hejlsberg** le créateur de :
 - **Turbo Pascal**
 - **.NET Framework**
 - **TypeScript** utilisé par le framework **Angular** de **Google**

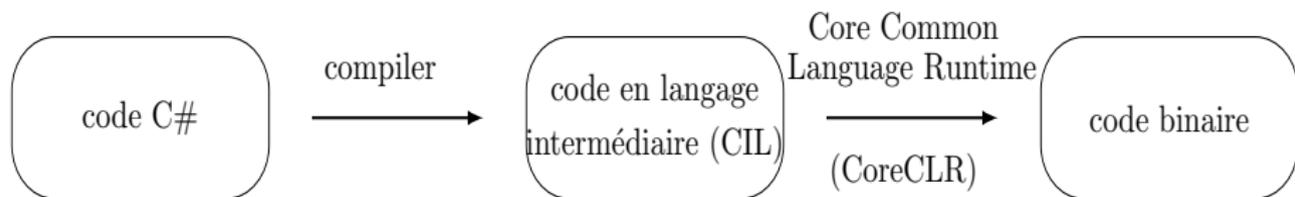
- Permettant de développer des applications dans l'environnement **.NET**.
- Permettant de développer des
 - Applications Web,
 - Applications Desktop (Client lourd),
 - Applications mobiles,
 - API REST,
 - Jeux,
 - ...

Comment ça fonctionne ?

- On écrit un programme **C#**.
- Le code **C#** sera transformé en langage intermédiaire (appelé **CIL** pour **Common Intermediate Language**).
- Le code CIL sera compilé par la machine virtuelle (appelée **CoreCLR** pour **Core Common Language Runtime**) pour avoir un code binaire.

Comment ça fonctionne ?

- On écrit un programme **C#**.
- Le code **C#** sera transformé en langage intermédiaire (appelé **CIL** pour **Common Intermediate Language**).
- Le code CIL sera compilé par la machine virtuelle (appelée **CoreCLR** pour **Core Common Language Runtime**) pour avoir un code binaire.



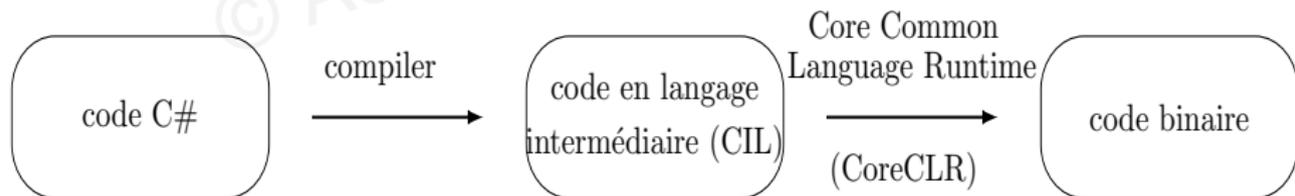
Code CIL vs Code binaire

- **CIL** : un code intermédiaire qu'on peut exécuter sur n'importe quelle machine.
- Code binaire : adapté à la machine sur laquelle il tourne.

© Achref EL M...

Code CIL vs Code binaire

- **CIL** : un code intermédiaire qu'on peut exécuter sur n'importe quelle machine.
- Code binaire : adapté à la machine sur laquelle il tourne.



CLR

Machine virtuelle de la plateforme **.NET Framework**

© Achref EL MOUËLMI

CLR

Machine virtuelle de la plateforme **.NET Framework**

CLR dispose aussi de

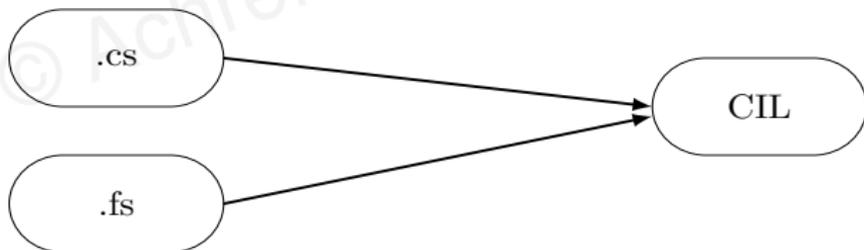
- **JIT (Just In Time)**
- **Garbage Collector**
- ...

CoreCLR ?

- Initialement une copie de **CLR** pour **.NET Core**
- Modifié par la suite pour supporté **Linux** et **Mac**
- **CLR** et **CoreCLR** maintenus séparément et en parallèle

Dans .NET Core

- On peut écrire un code **C#** et aussi **F#**.
- Tous ces langages seront compilés en code **CIL**.



C#

À partir d'un programme **C#**, il est possible de créer

- soit un fichier `.exe`
- soit une bibliothèque de classe sous la forme d'un fichier `.dll`

© Achref EL MOUËLHANI

C#

À partir d'un programme **C#**, il est possible de créer

- soit un fichier `.exe`
- soit une bibliothèque de classe sous la forme d'un fichier `.dll`

C'est quoi la différence ?

- `.exe` permet de lancer un programme
- `.dll` peut être utilisée par plusieurs programmes `.exe`

C#

À partir d'un programme **C#**, il est possible de créer

- soit un fichier `.exe`
- soit une bibliothèque de classe sous la forme d'un fichier `.dll`

C'est quoi la différence ?

- `.exe` permet de lancer un programme
- `.dll` peut être utilisée par plusieurs programmes `.exe`

Dans les deux cas

On parle d'un `assembly`

Quelques versions de C# (les plus importantes)

- **C# 12** (sortie en 2023) : spread operator (. .)
- **C# 11** (sortie en 2022) : propriétés obligatoires
- **C# 10** (sortie en 2021) : affectation et déclaration dans la dé-construction
- **C# 9** (sortie en 2020) : notion de `Record...`
- **C# 8** (sortie en 2019) : méthodes d'interface par défaut, `ReadOnly...`
- **C# 7** (sortie en 2016) : tuples, `out`, `ref...`
- **C# 6** (sortie en 2015) : interpolation de chaîne de caractères, indexeur...
- **C# 5** (sortie en 2012) : programmation asynchrone...
- **C# 4** (sortie en 2010) : typage dynamique...
- **C# 3** (sortie en 2007) : **LINQ...**
- **C# 2** (sortie en 2005) : généricité, nullable, itérateurs, types partiels...
- **C# 1** (sortie en 2002) : classes, interfaces, opérateurs, structures, instructions...

Principales versions d'.NET Core

- .NET 8.0 sorti en novembre 2023 (supportant C# 12)
- .NET 7.0 sorti en novembre 2022 (supportant C# 11)
- .NET 6.0 (LTS) sorti en novembre 2021 (supportant C# 10)
- .NET 5.0 sorti en novembre 2020 (supportant C# 9)
- .NET Core 3.1 (LTS) sorti en décembre 2019
- .NET Core 3.0 sorti en novembre 2019 (supportant C# 8)
- .NET Core 2.2 sorti en décembre 2018 (supportant C# 7.3)
- .NET Core 2.1 (LTS) sorti en mai 2018
- .NET Core 2.0 sorti en août 2017
- .NET Core 1.1 sorti en novembre 2016
- .NET Core 1.0 sorti en juin 2016

Remarque

- À partir de **.NET 5.0**, **.NET Core** a été rebaptisé **.NET**
- À partir de **.NET 5.0**, plus de nouvelles versions pour **.Net Framework**
- **ASP.NET Core** et **Entity Framework Core** suivent le même cycle de vie que **.NET Core**

Les règles de nommage en C#

- Pour les noms de projet et solution : **Le Pascal case**
- Pour les classes et les fichiers : **Le Pascal case**
- Pour les variables et les objets : **Le Camel case**
- Pour les méthodes : **Le Pascal case**

Les règles de nommage en C#

- Pour les noms de projet et solution : **Le Pascal case**
- Pour les classes et les fichiers : **Le Pascal case**
- Pour les variables et les objets : **Le Camel case**
- Pour les méthodes : **Le Pascal case**

Pour plus de détails

<https://wprock.fr/blog/conventions-nommage-programmation/>

Instructions

- Chaque instruction se termine par ;
- Il est possible d'écrire plusieurs instructions sur une même ligne (**mais** ce n'est pas une bonne pratique)

Comment organiser un projet C# ?

- Une classe par fichier.
- Possibilité de définir une classe dans plusieurs fichiers différents.
- Possibilité de regrouper plusieurs projets par solution.

Étapes

- Créer un nouveau projet `Fichier > Nouveau > Projet`
- Choisir C# dans `Tous les langages`
- Sélectionner `Application console` (à ne pas confondre avec `Application console (.NET Framework)`)



Étapes

- Cliquer sur Suivant
- Remplir les champs
 - Nom : **avec** ConsoleCore
 - Solution **avec** MaSolution
- Dans Emplacement, **Visual Studio** nous indique l'emplacement physique de notre projet (Par défaut dans `c:/utilisateurs/utilisateur/source/repos`).
- Valider

Pour connaître la version de **C#** utilisée dans le projet

- **Aller dans** Outils > Ligne de commande > Invite de commandes développeur
- **Dans la console, saisir la commande** `csc -langversion:?`

C#

Code obtenu

```
using System;

namespace ConsoleCore
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Syntaxe et concept assez proches de celles de **C++** et **Java**.

Explication

- On utilise `Using` pour importer les espaces de nom (comme en **C++**).
- Le mot-clé `namespace` permet d'indiquer l'espace de nom courant (comme un package **Java**).
- `static void Main(string[] args)` : point d'entrée de notre application console.

À partir de .NET 6, le modèle de projet pour les nouvelles applications de console C# génère le code suivant dans le fichier

Program.cs

```
Console.WriteLine("Hello, World!");
```

Pour exécuter, cliquez sur

-  ConsoleCore

- `Ctrl` + `F5`

- `fn` + `Ctrl` + `F5`

Dans la console

- Le résultat du programme est affiché.
- Le titre de la console indique l'emplacement de l'exécutable
`c:\utilisateurs\utilisateur\source\repos\MaSolution\ConsoleCore\bin\Debug\netcoreapp3.1\ConsoleCore.exe`

Pour accéder à l'exécutable (`ConsoleCore.exe`)

- Aller dans le menu **Affichage** et cliquer sur **Explorateur de solutions**
- Faire un clic droit sur **MaSolution** qui apparaît dans le panneau **Explorateur de solutions** et choisir **Ouvrir le dossier** dans l'**Explorateur de fichiers**
- Aller à `ConsoleCore\bin\Debug\netcoreapp3.1`
- Faire un double-clic sur `ConsoleCore.exe`

Constat

La console apparaît et disparaît très rapidement avant de voir `Hello world.`

© Achref EL M...

C#

Constat

La console apparaît et disparaît très rapidement avant de voir `Hello world.`

Solution

Ajouter une instruction bloquante

C#

Ajoutons l'instruction suivante puis exécutons le projet avant de vérifier l'exécutable

```
using System;

namespace ConsoleCore
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            Console.ReadKey();
        }
    }
}
```

Objectif

Écrire une solution **.NET** avec plusieurs langages de programmation.

Étape 1 : créer un projet F# dans MaSolution

- Dans l'Explorateur de solutions, faire clic droit sur MaSolution
- Aller à Ajouter > Nouveau projet
- Chercher Bibliothèque de classes



- Saisir ProjetFSharp dans Nom
- Valider
- Vérifier la présence de deux projets dans l'Explorateur de solutions

Code obtenu

```
namespace ProjetFSharp  
  
module Say =  
    let hello name =  
        printfn "Hello %s" name
```

© Achret

Code obtenu

```
namespace ProjetFSharp  
  
module Say =  
    let hello name =  
        printfn "Hello %s" name
```

Si on vérifie le répertoire `bin/Debug` du projet **ProjetFSharp**, on verra qu'il est vide (pas d'exécutable), donc inexploitable.

Étape 3 : générer le dll

- Aller dans l'Explorateur de solution
- Faire un clic droit sur le projet `ProjetFSharp`
- Choisir Générer

© Acti

Étape 3 : générer le dll

- Aller dans l'Explorateur de solution
- Faire un clic droit sur le projet `ProjetFSharp`
- Choisir Générer

Si on vérifie le répertoire `bin/Debug`, un fichier `.dll` a été généré.

Étape 4 : connecter les deux projets

- Aller dans l'Explorateur de solution
- Dans le projet `ConsoleCore`, Faire un clic droit sur Dépendances **et choisir** Ajouter une référence de projet
- Cliquer sur Projets **et cocher la case** `ProjetFSharp`
- Valider

Étape 4 : connecter les deux projets

- Aller dans l'Explorateur de solution
- Dans le projet `ConsoleCore`, Faire un clic droit sur Dépendances et choisir Ajouter une référence de projet
- Cliquer sur Projets et cocher la case `ProjetFSharp`
- Valider

Vérifier que `ProjetFSharp` figure dans la liste de dépendances de `ConsoleCore`.

Étape 5 : utilisons la classe `Say` écrite en *F#* dans *C#*

```
using System;
using ProjetFSharp;
namespace ConsoleCore
{
    class Program
    {
        static void Main(string[] args)
        {
            Say.hello("wick");
            Console.WriteLine("Hello World!");
        }
    }
}
```

En exécutant, le résultat est

```
Hello wick  
Hello World!
```

Pour écrire dans la console

```
Console.WriteLine("Un message et un retour à la ligne");
```

Pour écrire sans retourner à la ligne

```
Console.Write("Un message sans retour à la ligne");
```

Commentaire sur une seule ligne

```
// commentaire
```

© Achref EL MOUELHI

Commentaire sur une seule ligne

```
// commentaire
```

Raccourcis Visual Studio 2019

- Pour commenter : `Ctrl` + `k` puis `Ctrl` + `c`
- Pour décommenter : `Ctrl` + `k` puis `Ctrl` + `u`

Commentaire sur une plusieurs lignes (Raccourci VS :)

```
/* le commentaire  
   la suite  
   et encore la suite  
*/
```

© Achref EL W.

Commentaire sur une plusieurs lignes (Raccourci VS :)

```
/* le commentaire  
   la suite  
   et encore la suite  
*/
```

Raccourci Visual Studio 2019

Pour commenter ou décommenter : Ctrl + Shift + /

Commentaire pour la documentation

```
/// un commentaire qui sera inclus dans la  
documentation
```

© Achref EL MOUL

Commentaire pour la documentation

```
/// un commentaire qui sera inclus dans la  
documentation
```

Pour le générer avec Visual Studio 2019

Aller dans Edition > IntelliSense > Insérer un
commentaire

Modifier la console

- `Console.BackgroundColor = ConsoleColor.Red;` pour mettre la couleur de fond en rouge
- `Console.ForegroundColor = ConsoleColor.Yellow;` pour mettre la couleur de caractères en jaune
- `Console.ResetColor();` pour réinitialiser les couleurs
- `Console.Clear();` pour effacer le contenu de la console
- `Console.SetCursorPosition(50, 50);` pour positionner la console
- ...

La documentation officielle (en français)

<https://docs.microsoft.com/fr-fr/dotnet/csharp/index>