Chapitre 4
Mutualisation de code

1. Introduction

L'intérêt principal de Xamarin réside dans la possibilité de partager du code entre plusieurs plateformes mobiles.

Plusieurs méthodes de partage de code existent et nous verrons dans ce chapitre chaque stratégie avec ses avantages et inconvénients.

Ensuite, nous verrons comment réaliser une injection de dépendance simple. Enfin, nous verrons l'architecture MVVM (*Model View ViewModel*) idéale pour mutualiser un maximum de code.

2. Quelles sont les possibilités pour partager son code en Xamarin ?

2.1 Présentation

Deux possibilités s'offrent à nous pour partager du code dans chacun de nos projets :

– Utiliser un SharedProject.

– Utiliser une Portable Class Library (PCL).

2.2 SharedProject

2.2.1 Présentation

Un SharedProject, comme son nom l'indique est un type de projet permettant de simplifier le partage de fichier entre projets.

Le SharedProject n'est pas une bibliothèque de classe, il ne produit aucun binaire. Lors de la compilation d'un des projets qui le référencent, les fichiers du SharedProject sont rajoutés au projet compilé.

Un SharedProject n'est jamais compilé tout seul, il existe uniquement pour regrouper des fichiers de code source et être inclus facilement dans un autre projet. Il ne peut pas avoir des références propres, mais il a accès aux références de tous les autres projets qui l'utilisent.

Par exemple

Si vous ajoutez une référence à MVVMLight sur vos projets iOS et Android, le SharedProject partagé entre ces deux OS aura aussi accès à MVVMLight.

Si dans le projet partagé vous utilisez MVVMLight et que l'un des projets qui l'appellent n'a pas référencé MVVMLight, ce dernier ne compilera pas.

Le grand avantage des SharedProjects est qu'ils supportent l'utilisation des directives de précompilation.

2.2.2 Avantage

Cette approche est très simple à utiliser. Elle autorise aussi l'utilisation des directives préprocesseur (voir la section Directive de préprocesseur).

2.2.3 Inconvénients

On ne peut pas distribuer un binaire de ce projet. Il sert uniquement au partage de code dans une solution que l'on maîtrise.

Il est difficile de partager un SharedProject entre plusieurs applications.

2.2.4 Création d'un SharedProject

Sous Visual Studio

- Aller dans l'explorateur de solution.
- Faire un clic droit sur un dossier ou la solution.
- Sélectionner **Ajouter - Ajouter un nouveau projet**.
- Sélectionner **SharedProject**.
- Ajouter aux projets mobiles la référence au SharedProject créé.

Sous Xamarin Studio

- Aller dans l'explorateur de solution.
- Faire un clic droit sur la solution.
- Sélectionner **Add - Add New Project**.
- Sélectionner **Library**, puis sélectionner **SharedProject**.
- Ajouter aux projets mobiles la référence au SharedProject créé.

2.2.5 Directive préprocesseur

Quand vous utilisez le partage de code via les SharedProject, vous avez la possibilité d'utiliser des directives préprocesseur.

Ces directives préprocesseur vous permettent d'inclure ou d'exclure du code lors de la compilation vers une plateforme spécifique.

Exemple d'utilisation de directive préprocesseur

```
#if __ANDROID__
//Si on est sur Android
Console.WriteLine("Hello Android");
#elif __IOS__
//Sinon si on est sur iOS
Console.WriteLine("Hello iPhone and iPad");
#elif WINDOWS_UWP
//Sinon si on est sur Windows 10
Console.WriteLine("Hello Windows 10 apps");
#endif
```

Xamarin définit par défaut un grand nombre de variables préprocesseur :

– `__MOBILE__` : si c'est compilé sur iOS ou Android.
– `__IOS__` : identifie un projet pour iOS.
– `__TVOS__` : identifie un projet pour Apple TV.
– `__WATCHOS__` : identifie un projet pour les montres Apple Watch.
– `__ANDROID__` : identifie un projet Android.

Remarque

Pour Android, Xamarin a créé des directives spécifiques pour du code ne s'exécutant qu'à partir d'un niveau minimum d'API. Par exemple, `__ANDROID__11__` permettra de compiler le code si le projet est configuré avec comme API minimum `11`.

Exemple de code pour lancer un appel téléphonique

```
public void CallPhone(string phoneNumber)
{
      #if __ANDROID__
      //Si on est sur Android
      var strUri = string.Format("tel:{0}", phoneNumber);
      var uri = global::Android.Net.Uri.Parse(strUri);
      var intent = new Intent(Intent.ActionDial, uri);
      intent.AddFlags(ActivityFlags.NewTask);
      Application.Context.StartActivity(intent);

      #elif __IOS__
      //Si on est sur iOS
      var url = new NSUrl("tel :" + phoneNumber);
      UIApplication.SharedApplication.OpenUrl(url);

      #elif WINDOWS_UWP
      // Si on est sur Windows 10
      //On utilise l'applis Skype pour lancer un appel
      Uri uri = new Uri(string.Format("skype:{0} ?call", phoneNumber));
      Windows.System.Launcher.LaunchUriAsync(uri);
      #endif
}
```

Remarque

Vous pouvez créer vos propres directives préprocesseur grâce à la notation suivante : `#define NOMDELADIRECTIVE` ou directement dans les options de compilation de Visual Studio et Xamarin Studio.

Utiliser des directives préprocesseur pour le développement cross-plateforme n'est pas le moyen le plus propre. Elles peuvent suffire pour des cas simples. Plus les méthodes sont compliquées, plus la syntaxe `#if / #elif` est difficile à lire et à maintenir ; il faut donc l'utiliser avec modération.

2.3 Portable Class Library

2.3.1 Présentation

Une PCL (*Portable Class Library*) est un sous-ensemble du Framework .NET qui est compatible avec plusieurs plateformes (iOS, Android, Windows, Silverlight, Xbox...). Lorsque vous créez une PCL, vous pouvez choisir avec quelles plateformes celle-ci est compatible. Les PCL ont pour but de simplifier le développement de bibliothèques de classes entre les différentes versions du Framework .NET.

La PCL étant compilée une fois, on ne peut pas utiliser de directive de préprocesseur spécifique à une plateforme. Cela rend difficile l'utilisation de code spécifique à une plateforme. Néanmoins, grâce à l'injection de dépendance, que l'on abordera juste après, on peut facilement passer outre.

Remarque

La plupart des bibliothèques de classes populaires sont des PCL : Newtownsoft.Json, MVVMLight, MvvmCross...

2.3.2 Avantages

L'un des gros avantages de la PCL est de centraliser le partage de code dans un seul et unique projet. Ce projet peut être facilement partagé à d'autres applications contrairement au SharedProject. On peut facilement partager ce type de projet en partageant la dll produite.

2.3.3 Inconvénients

On ne peut pas utiliser du code spécifique à une plateforme (en utilisant des directives de préprocesseur) car la PCL est configurée pour fonctionner sur toutes les plateformes.

2.3.4 Création d'une PCL

Sous Visual Studio

- Aller dans l'explorateur de solution.
- Faire un clic droit sur un dossier ou la solution.
- Sélectionner **Ajout - Ajouter un Nouveau Projet**.
- Sélectionner **Bibliothèque de classe** (**Portable pour iOS, Android et Windows**).
- Ajouter aux projets mobiles la référence à votre PCL.

Sous Xamarin Studio

- Aller dans l'explorateur de solution.
- Faire un clic droit sur la solution.
- Sélectionner **Add - Add New Project**.
- Sélectionner **Library**, puis **Portable Library**.
- Ajouter aux projets mobiles la référence à votre PCL.