UWP 開発メモ

[Universal Windows Platform]

Windows SDK

・https://developer.microsoft.com/ja-jp/windows/downloads/windows-10-sdk

Windows Template Studio

・https://codezine.jp/article/detail/11282

インストール

1. メニュー［ツール］－［拡張機能と更新プログラム］
2. ダイアログの左側で［オンライン］を選択し、右上の検索窓に「"Windows Template Studio"」と入力、検索すると見つかる
3.［ダウンロード］ボタンをクリック

Windows Community Toolkit

・https://docs.microsoft.com/ja-jp/windows/communitytoolkit/

インストール

1.Visual Studio 2017 以上
2.Build 16299 以上のプロジェクト
3.NuGet パッケージの管理から、Microsoft.Toolkit.UWP を検索し必要なパッケージをインストール

参照

■ XAML

```
xmlns:controls="using:Microsoft.Toolkit.Uwp.UI.Controls"
```

■ C#

```
using Microsoft.Toolkit.Uwp;
```

Entity Framework Core

・https://docs.microsoft.com/ja-jp/ef/core/get-started/uwp/getting-started
・https://docs.microsoft.com/ja-jp/ef/core/get-started/netcore/new-db-sqlite
・http://kuttsun.blogspot.com/2018/01/entity-framework-core.html

インストール

1. NuGet パッケージマネージャー Microsoft.EntityFrameworkCore.Sqlite

Sqlite3 ファイルが生成される場所

・ApplicationData.Current.LocalFolder.Path

ソリューションの構成

1.UWP プロジェクト (Universal Windows)
2.Model プロジェクト (.NETStarndard Library + EntityFrameworkCore.Sqlite + EntityFrameworkCore.Tools)
3. 移行プロジェクト (ダミー)(.NETCore Console)

移行

・https://docs.microsoft.com/ja-jp/ef/core/managing-schemas/migrations/index
・http://typea.info/blg/glob/2017/08/uwp-sqlite-entityframework-core.html

■パッケージマネージャーコンソールから以下を実行

1. プロジェクトの構成
    1.DbContext のサブクラス
    2.DbSet< モデル >
    3.context.Database.Migration() の呼び出し記述
2. 初期マイグレーションの構成
    1. 移行プロジェクト (ダミー) をスタートアッププロジェクトに設定
    2. 既定のプロジェクトに、Model プロジェクトを指定
    3.PM> Add-Migration { 初期 Migration 名 }
    4.Model プロジェクトの Migratinos ディレクトリに DbContext のリフレクションにより移行スクリプトが構成される
3.UWP アプリの実行
    1.context.Database.Migration() の呼び出し
    2.Sqlite データベースが作成、マイグレーション実行
4. データベースの確認
    1. select * from __EFMigrationsHistory にマイグレーション情報
5. 変更マイグレーション
    1. モデルの追加・変更などを Model プロジェクトに適用
    2. 移行プロジェクト (ダミー) をスタートアッププロジェクトに設定
    3. 既定のプロジェクトに、Model プロジェクトを指定
    4.PM> Add-Migration { 変更 Migration 名 }
    5.Model プロジェクトの Migratinos ディレクトリに DbContext のリフレクションにより移行スクリプトが構成される
6. 以下繰り返し

Powershell コマンド

■初期移行

```
Add-Migration {任意のマイグレーション名 }
```

・ディレクトリの下で 3 つのファイルがプロジェクトに追加

ファイル	内容
00000000000000_{任意のマイグレーション名 }.cs	メインの移行ファイル。 (Up() で) 移行を適用し、(Down() で) それを元に戻すために必要な操作が含まれます。

00000000000000_｛任意のマイグレーション名}.Designer.cs	移行メタデータ ファイル。 EF によって使用される情報が含まれます。
MyContextModelSnapshot.cs	現在のモデルのスナップショット。 次の移行を追加するときの変更内容の決定に使用されます。

■データベースを更新

```
Update-Database
```

■移行の削除

・移行の追加後、適用する前に EF Core モデルの追加変更が必要なことに気付く場合があります。 最後の移行を削除するには、このコマンドを使用

```
Remove-Migration
```

■移行を元に戻す

・移行をデータベースに既に適用しているが、元に戻す必要がある場合、同じコマンドを使用して移行を適用できますが、ロールバックする移行の名前を指定

```
Update-Database { ロールバックする移行の名前 }
```

■ SQL スクリプトを作成

```
Script-Migration
```

■実行時に移行を適用

・起動中または最初の実行中、実行時に移行を適用するアプリがあります。 Migrate() メソッドを使用してこれを行います

```
myDbContext.Database.Migrate();
```

MVVM Light

■ RelayCommand

・http://my-clip-devdiary.blogspot.com/2011/01/relaycommand-mvvm-light-toolkit.html

コマンドパラメータなし

・XAML

```
<Button Content="Add"  Command="{x:Bind ViewModel.AddCommand}" />
```

・ViewModel

```
public RelayCommand AddCommand { get; private set; }
    = new RelayCommand(async () => {
        await new MessageDialog("Foo", "Bar").ShowAsync();
    });
```