# DÉPLOYEZ VOTRE APPLICATION .NET AVEC CLICKONCE

Samuel Boïté <thulemalta@gmail.com>

csharp .NET Framework deployment

GitBook

# Table of Contents

Le déploiement est un des points clefs du développement d'une application. C'est le processus permettant la mise à disposition d'une application à des utilisateurs.

ClickOnce est un framework de déploiement créé par Microsoft en 2005. Il permet aux utilisateurs d'installer une application par un simple clic sur une page web. Il est disponible depuis la sortie du Framework .NET 2.0.
Il fonctionne un peu comme le « Java Web Start » de Java.

ClickOnce offre l'énorme avantage de permettre la mise à jour automatique et intelligente des applications déployées. Lorsque l'utilisateur lance une application ClickOnce, cette dernière vérifie si une nouvelle mise à jour est disponible ; si oui elle propose son installation.
Il est adapté à applications relativement petites. Ne vous attendez donc pas à le voir faire des miracles avec des jeux XNA ou des grosses applications. 🌟

ClickOnce est utilisable dans tous les langages utilisant le framework .NET. Nous utiliserons dans ce tutoriel un assistant de déploiement propre au VB.NET et au C#. Si vous comptez déployer du Delphi ou du F# par exemple, l'installation sera plus compliquée.
L'assistant publication, que j'utilise ici, est disponible depuis la version 2005 de Visual Studio. Dans ce tutoriel, j'utilise Visual Studio Community 2015, qui est gratuit.

## Remerciements

Merci à artragis pour ses conseils lors de la beta.
Le logo du tutoriel provient du *Tango Pack* (licence *Creative Commons* BY-NC 3.0 NL)

# Mise en place

Nous allons voir ici comment déployer un logiciel avec ClickOnce. Nous n'utiliserons que les fonctionnalités « simples » de l'outil et ne nous attarderons pas à apprendre comment l'utiliser de manière avancée.

ClickOnce est bien fait. :magicien:
Vous pouvez distribuer votre application ClickOnce *via* beaucoup de supports, à savoir un **répertoire en réseau**, un **site web** ou même un **CD-ROM** ou un **DVD** (on les utilise encore ? 😯 ).

Vous avez donc l'embarras du choix. Une fois que vous avez décidé quel support utiliser, passez à la suite du tutoriel. 😄 Vous pouvez même répéter l'opération afin de publier l'application sur plusieurs supports.

Il me semble nécessaire de définir quelques points de vocabulaire :

- Quand je parlerai de **logiciel**, ce sera **notre** logiciel, celui qu'on a codé et que l'on veut déployer ;
- Quand je parlerai d'**application**, ça sera l'application ClickOnce, celle que l'on configurera ici.

# Publication de l'application ClickOnce

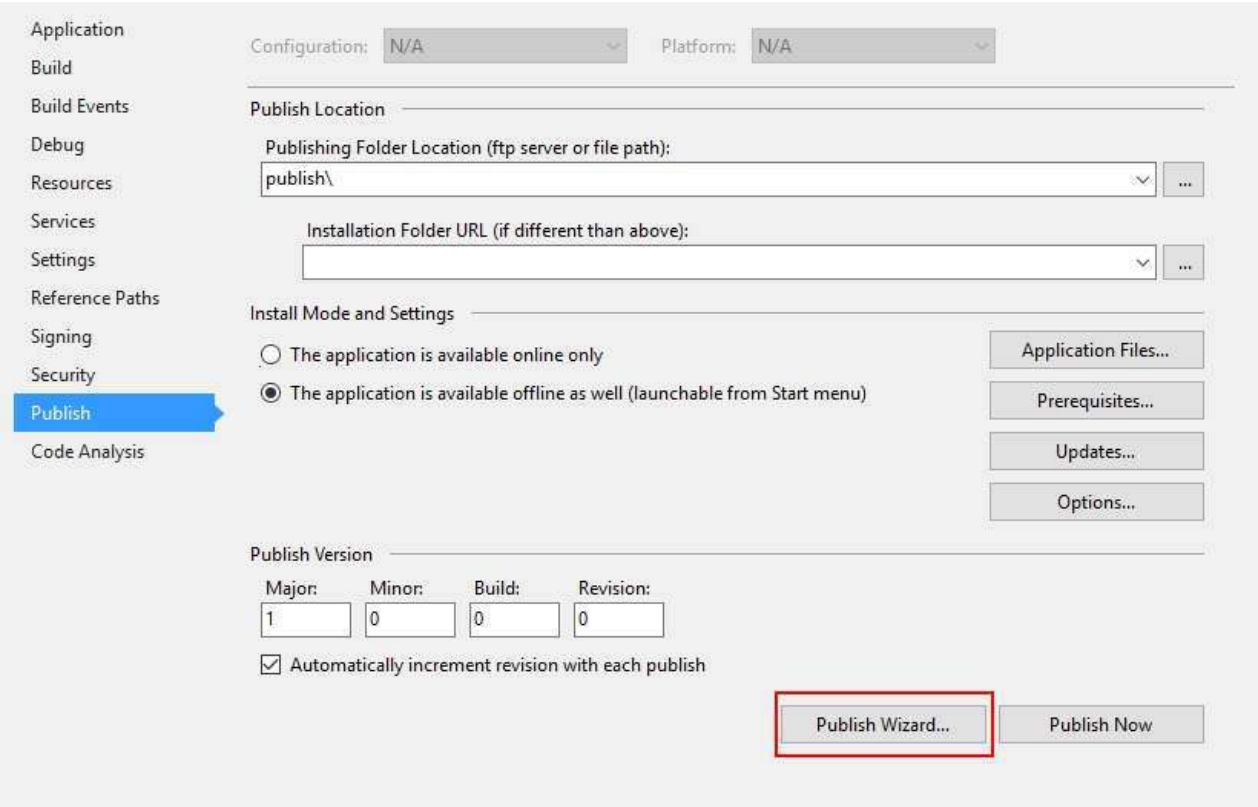
Après un travail acharné, j'ai réalisé un petit logiciel de test, afin de vous vous montrer comment utiliser ClickOnce :


---

Allez, c'est parti. Nous allons publier notre logiciel avec ClickOnce. Pour ce faire, trois moyens :

- Utiliser **l'assistant publication** est la manière la plus simple, et par conséquent celle que nous explorerons ici ;
- La page **« publier »** des paramètres du projet nous permet d'arriver au même résultat ;
- On peut enfin créer les fichiers **à la main**, mais je ne suis pas masochiste.

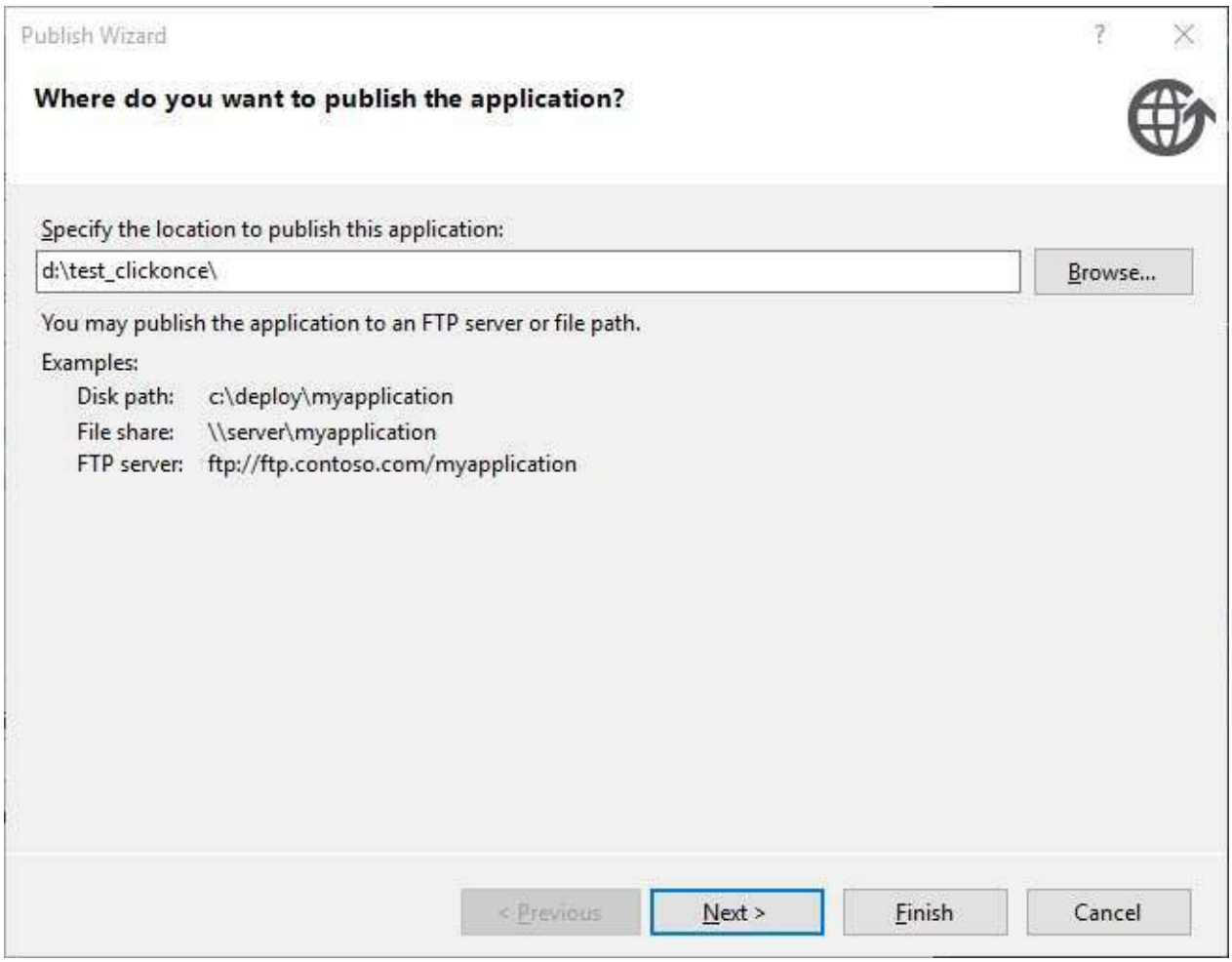
Nous allons donc utiliser l'**assistant publication**. Pour y accéder, allez dans l'**explorateur de solutions** (*Solution Explorer*), **cliquez-droit** sur votre **projet** puis ouvrez ses **paramètres**. Allez dans l'onglet *Publish* et lancez l'**assistant publication** (*Publish Wizard*).




Dans la page 1, on vous demandera de renseigner le répertoire de publication de l'application. C'est ici que seront enregistrés les fichiers de votre application ClickOnce. On y trouvera aussi une page web de téléchargement très basique, servant d'exemple.
Vous avez plusieurs choix de répertoires :

- **un répertoire sur un disque local** comme `C:\publication` par exemple ;
- **un répertoire en réseau**, sous la forme `\\NOM_DU_SERVEUR\publication` ou `\\192.168.0.3\publication` par exemple ;
- **un répertoire FTP**, comme `ftp://192.168.0.3/publication`.

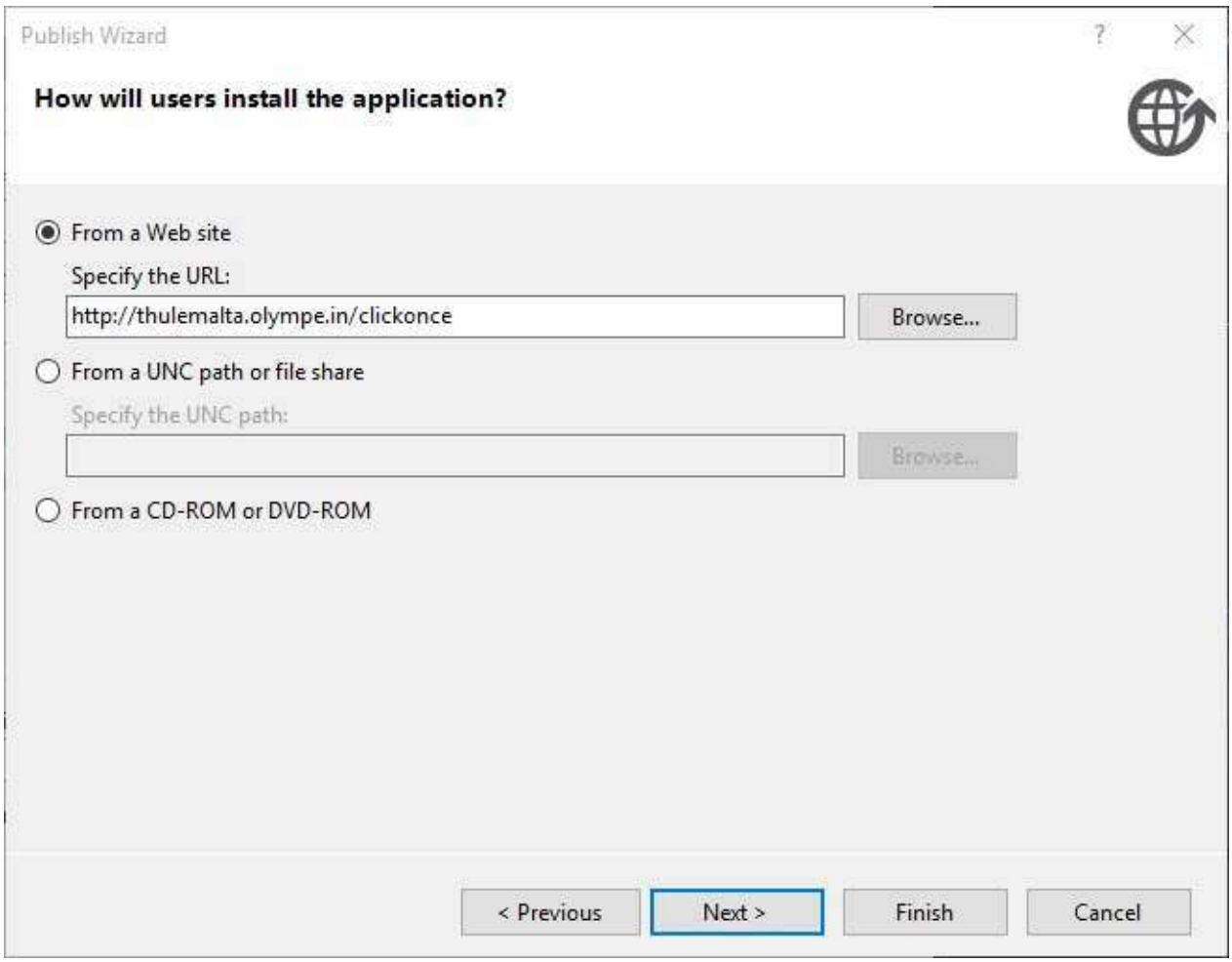
[[attention]] | Je le répète : ce chemin n'est pas celui qu'utiliseront les utilisateurs pour récupérer votre application. C'est juste un répertoire où l'installeur sera enregistré.




C'est dans la page 2 que vous spécifierez par quel moyen vous distribuerez l'application ClickOnce.

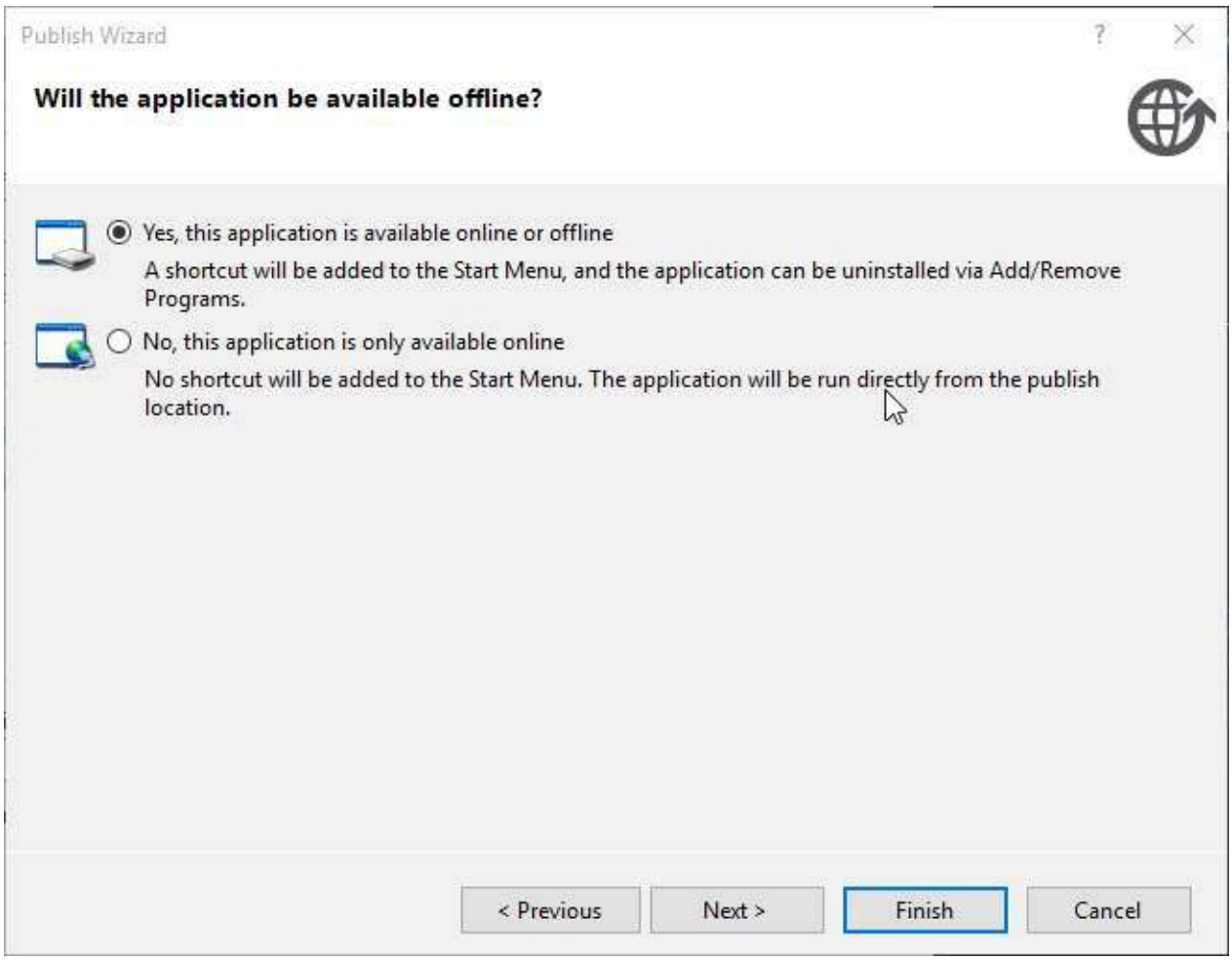
Vous avez cette fois le choix entre :

- **un site internet** : spécifiez dans ce cas l'URL de la page à partir de laquelle sera téléchargée l'application, comme `http://mon_application_clickonce.com/téléchargement` par exemple ;
- **un chemin UNC**, c'est-à-dire un répertoire en réseau, sous la forme `\\NOM_DU_SERVEUR\publication` ou `\\192.168.0.3\publication` par exemple ;
- **un CD-ROM** ou un **DVD**.




Dans la page 3 vous devrez choisir si cette application sera **en ligne** ou **hors ligne**. Une application en ligne télécharge le logiciel à chaque lancement et requiert donc une connexion internet pour fonctionner. Une application hors-ligne sauvegarde le logiciel sur le disque dur de l'utilisateur, et l'affiche dans le menu Démarrer.

Si vous avez choisi CD-ROM ou DVD à la page précédente, vous devrez aussi spécifier une adresse web depuis laquelle l'application sera en ligne. Le disque récupérera le logiciel depuis cette adresse.
Vous pouvez bien-sûr choisir de la rendre complètement hors-ligne également. Dans ce cas, vous ne pourrez pas distribuer de mises à jour.




C'est terminé !

Appuyez sur « Terminer » et votre programme d'installation ClickOnce sera sauvegardé, à l'emplacement spécifié précédemment. Génial, non ? 😄

Dans la partie suivante, nous verrons comment mettre l'installeur ClickOnce à disposition des utilisateurs.

# Mise à disposition de l'application ClickOnce

Dans la partie précédente, nous avons généré les fichiers de notre application ClickOnce. Explorons-les.

```
|   publish.htm
|   setup.exe
|   Test_ClickOnce.application
|
└───Application Files
    └───Test_ClickOnce_1_0_0_0
            Test_ClickOnce.application
            Test_ClickOnce.exe.config.deploy
            Test_ClickOnce.exe.deploy
            Test_ClickOnce.exe.manifest
```

Regardons à quoi servent chacun de ces fichiers :

- `publish.htm` : la page web à partir de laquelle vos utilisateurs téléchargeront l'application. Vous pouvez la personnaliser à votre guise si vous la trouvez, comme moi, assez moche ;
- `setup.exe` : l'installeur des pré-requis. Par défaut, il s'agit de la version du .NET Framework que vous utilisez. Si vous regardez la page `publish`, vous remarquerez qu'elle est liée à cet installeur ;
- `Test_ClickOnce.application` : le manifeste de notre application ClickOnce. C'est lui qui sera téléchargé et lancé par l'utilisateur ;
- Sous-dossiers de `Application Files` : toutes les versions du logiciel publié.

Si vous voulez tester, vous pouvez lancer l'application ClickOnce. Ça ressemble à ça :

!(https://www.youtube.com/watch?v=V9NDwvStSsY)

Pour la suite, c'est simple:

- Si vous voulez rendre votre application disponible depuis un dossier partagé, copiez l'installeur ClickOnce à cet endroit ;
- Si vous voulez la mettre en ligne, uploadez tout le répertoire et faites un lien vers l'installeur ClickOnce ;
- Enfin, si utilisez encore les CD-ROM/DVD, gravez le répertoire dessus. Vous pouvez faire que l'installeur se lance automatiquement lors de l'insertion du disque (voir la section « Paramètres basiques » de cette partie).

Si vous utilisez un serveur non-IIS pour distribuer votre application, n'oubliez pas de rajouter les types MIME liés à ClickOnce ; le cas échéant votre application ne se lancera pas en « un seul clic ». Il vous faudra ajouter ces types.

# Paramètres basiques

Vous vous souvenez du déploiement de notre première application ? Nous étions allé dans les paramètres du projet, puis nous étions allé dans l'onglet *Publish* des paramètres. Eh bien c'est exactement là que se trouvent les paramètres de l'application.

Rendez-vous à cet endroit, et différentes options s'offriront à vous :

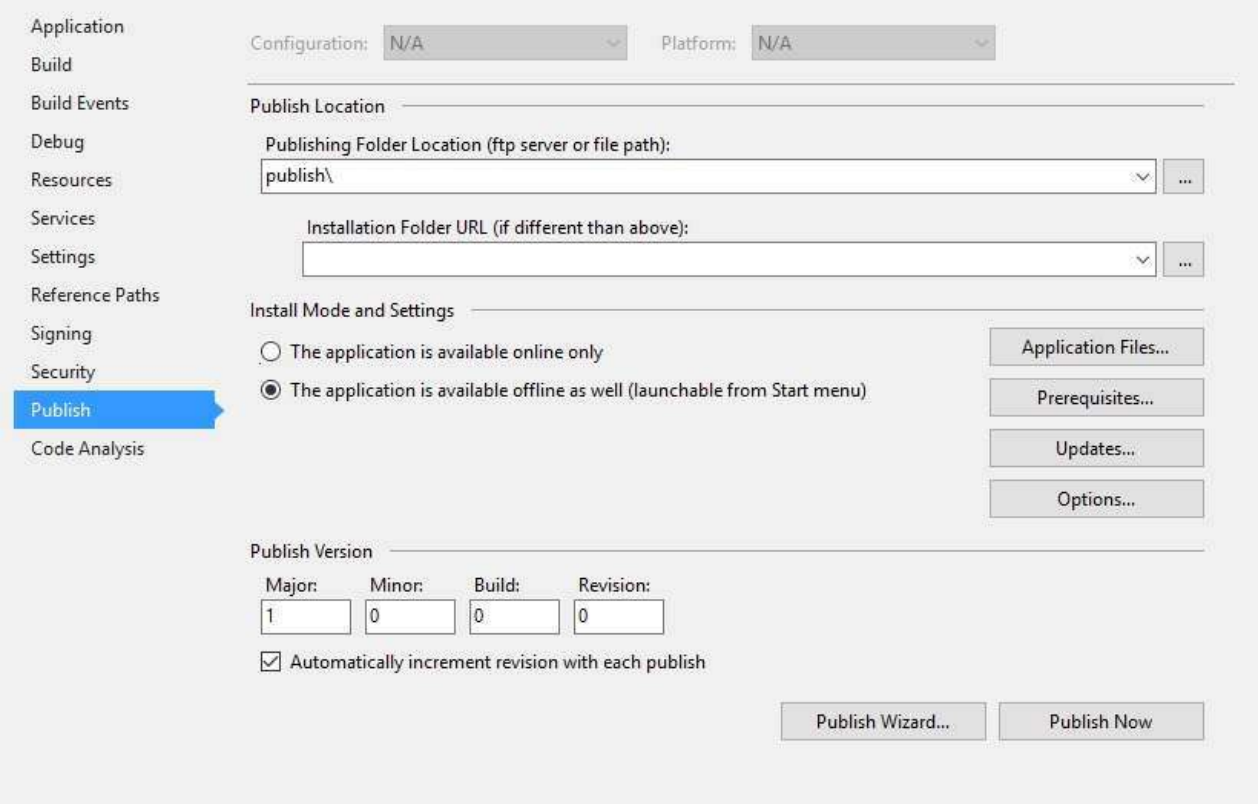
- En haut, l'emplacement des **fichiers de publication** (*Publishing Folder Location*) ;
- Juste en dessous, l'**URL** où l'utilisateur **téléchargera l'application** ;
- Et enfin, si l'application est **en ligne**/**hors ligne**.

Ces paramètres peuvent vous permettre de corriger les éventuelles erreurs que vous avez commises dans l'assistant publication.

# Gestion des mises à jour

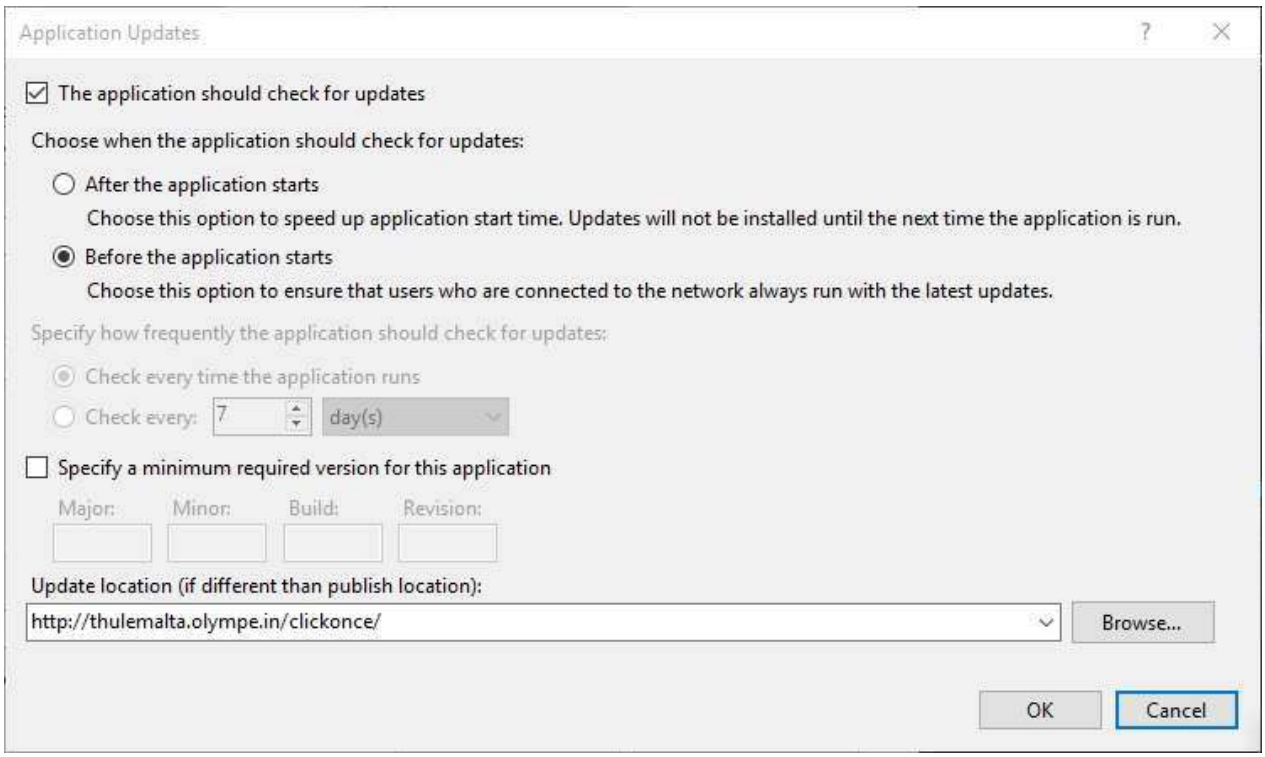
Nous allons nous atteler à la gestion des mises à jour dans ClickOnce.
En plus d'être très bien gérées (ClickOnce ne télécharge que les fichiers mis à jour par exemple) les mises à jour sont très simples à déployer.

Développez votre mise à jour, revenez dans le menu *Publish*, choisissez la nouvelle version de votre application puis cliquez sur le bouton *Publish Now*. Les fichiers ClickOnce seront mis à jour et vous devrez les *uploader* à nouveau sur votre site.
Si on les regarde de nouveau, on se rend compte qu'un nouveau dossier est apparu dans notre répertoire « Application Files », contenant les nouveaux fichiers de notre version.




Oui. C'est tout. C'était assez simple, non ? 😄

Vous pouvez aussi choisir quand et comment les applications sont mises à jour. Pour ce faire, ouvrez la boite de dialogue *Updates* et vous pourrez paramétrer quand l'application les recherche. Et si vous avez le souci de l'obsolescence, vous pouvez aussi choisir la version minimum requise pour pouvoir effectuer les mises à jour.


Application Updates
The application should check for updates
Choose when the application should check for updates:
After the application starts
Choose this option to speed up application start time. Updates will not be installed until the next time the application is run.
Before the application starts
Choose this option to ensure that users who are connected to the network always run with the latest updates.
Specify how frequently the application should check for updates:
Check every time the application runs
Check every:
7
day(s)
Specify a minimum required version for this application
Major:
Minor:
Build:
Revision:
Update location (if different than publish location):
http://thulemalta.olympe.in/clickonce/
Browse...
OK
Cancel

# Conclusion

C'est bon ! Vous êtes fin prêts à déployer votre application sur ClickOnce. 😄
Cette partie du tutoriel suffira à bon nombre de personnes, mais si vous voulez encore plus (signer votre manifeste, créer votre propre installeur ClickOnce, définir des stratégies de mise à jour, etc) la partie suivante est pour vous.

# Utilisation avancée

Dans cette partie, nous explorerons plus en profondeur les différentes options de personnalisation de ClickOnce qui s'offrent à nous.

# Support de différentes versions du .NET Framework

Il est assez fréquent de créer des applications supportant plusieurs versions du .NET Framework.
Or, pour que cette configuration marche sur une application ClickOnce, il faut éditer le fichier manifeste de notre application ClickOnce.

Le manifeste, c'est *grosso modo* le fichier de configuration d'une application ClickOnce. Écrit en XML, il contient les stratégies de mise à jour, les stratégies de sécurité et les *hash* de tous les fichiers pour déterminer les fichiers qui ont changé lors des mises à jour.

Il se situe à la racine de notre dossier de déploiement et a pour extension `.application`.
Ouvrez le avec votre éditeur XML favori et cherchez la balise `<compatibleFrameworks />`.

Vous devriez voir quelque chose comme ça :

```
<compatibleFrameworks xmlns="urn:schemas-microsoft-com:clickonce.v2">
  <framework targetVersion="4.6" profile="Full" supportedRuntime="4.0.30319" />
</compatibleFrameworks>
```

Ajoutez entre ces deux balises les codes XML des versions que vous voulez supporter. Voici un tableau récapitulatif des XML à ajouter en fonction des versions les plus récentes du .NET Framework :

| Version du .NET Framework | XML correspondant |
|---|---|
| 4.6 | `<framework targetVersion="4.6" profile="Full" supportedRuntime="4.0.30319" />` |
| 4.5.2 | `<framework targetVersion="4.5.2" profile="Full" supportedRuntime="4.0.30319" />` |
| 4.5.1 | `<framework targetVersion="4.5.1" profile="Full" supportedRuntime="4.0.30319" />` |
| 4.5 | `<framework targetVersion="4.5" profile="Full" supportedRuntime="4.0.30319" />` |
| 4 Client | `<framework targetVersion="4.0" profile="Client" supportedRuntime="4.0.30319" />` |
| 4 Full | `<framework targetVersion="4.0" profile="Full" supportedRuntime="4.0.30319" />` |
| 3.5 Client | `<framework targetVersion="3.5" profile="Client" supportedRuntime="2.0.50727" />` |
| 3.5 Full | `<framework targetVersion="3.5" profile="Full" supportedRuntime="2.0.50727" />` |
| 3.0 | `<framework targetVersion="3.0" supportedRuntime="2.0.50727" />` |

| Version du .NET Framework | Inclus par défaut dans... | Installable dans... |
|---|---|---|
| .NET 4.6 | 10 | Vista (et ultérieur) |
| 4.5.2 | - | Vista (et ultérieur) |
| 4.5.1 | 8.1 | Vista (et ultérieur) |
| 4.5 | 8 | Vista (et ultérieur) |
| 4 | - | Vista (et ultérieur) |
| 3.5 | 10, 8.1, 8, 7 | Vista (et ultérieur) |
| 3.0 | Vista | Vista (et ultérieur) |

Ensuite, il vous faudra éditer le fichier `App.config` de votre projet pour ajouter les mêmes versions. C'est une nouvelle fois un fichier XML, cherchez la balise `<startup />` cette fois. Vous devriez avoir quelque chose comme ceci :

```
<startup>
  <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6"/>
</startup>
```

Faites comme la dernière fois. Ajoutez le code XML des versions du .NET Framework que vous souhaitez supporter, j'ai re-fait un tableau récapitulatif ci-dessous :

| Version du .NET Framework | XML correspondant |
|---|---|
| 4.6 | `<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6"/>` |
| 4.5.2 | `<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2"/>` |
| 4.5.1 | `<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.1"/>` |
| 4.5 | `<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5"/>` |
| 4 Client | `<supportedRuntime version="v4.0.30319" sku=".NETFramework,Version=v4.0,Profile=Client" />` |
| 4 Full | `<supportedRuntime version="v4.0.30319" sku=".NETFramework,Version=v4.0" />` |
| 3.5 Client | `<supportedRuntime version="v2.0.50727" sku="Client"/>` |
| 3.5 Full | `<supportedRuntime version="v2.0.50727"/>` |
| 3.0 | `<supportedRuntime version="v2.0.50727"/>` |

Une fois que c'est terminé, ré-ouvrez le fichier manifeste de notre application ClickOnce.
Cherchez cette fois la balise `<dependency />`.
Entre les deux balises, insérez le code suivant :

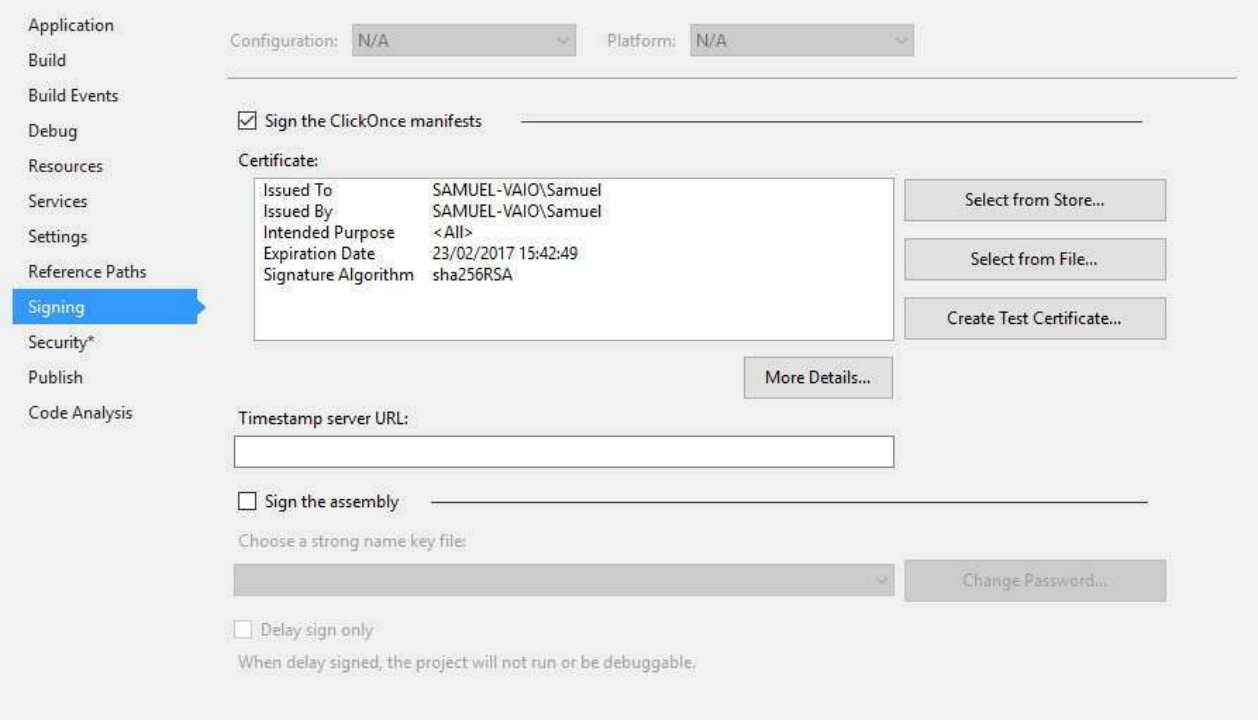
```
<dependentAssembly dependencyType="preRequisite" allowDelayedBinding="true">
  <assemblyIdentity name="Microsoft.Windows.CommonLanguageRuntime" version="<!-- versi
on du CLR ici -->" />
</dependentAssembly>
```

Et enfin, ajoutez la plus petite version commune du CLR (par exemple, si vous voulez supporter le .NET Framework 4.0 et 3.5, vous n'aurez pas le choix, il faudra ajouter la version `2.0.50727` du CLR.

# Signature de l'application ClickOnce

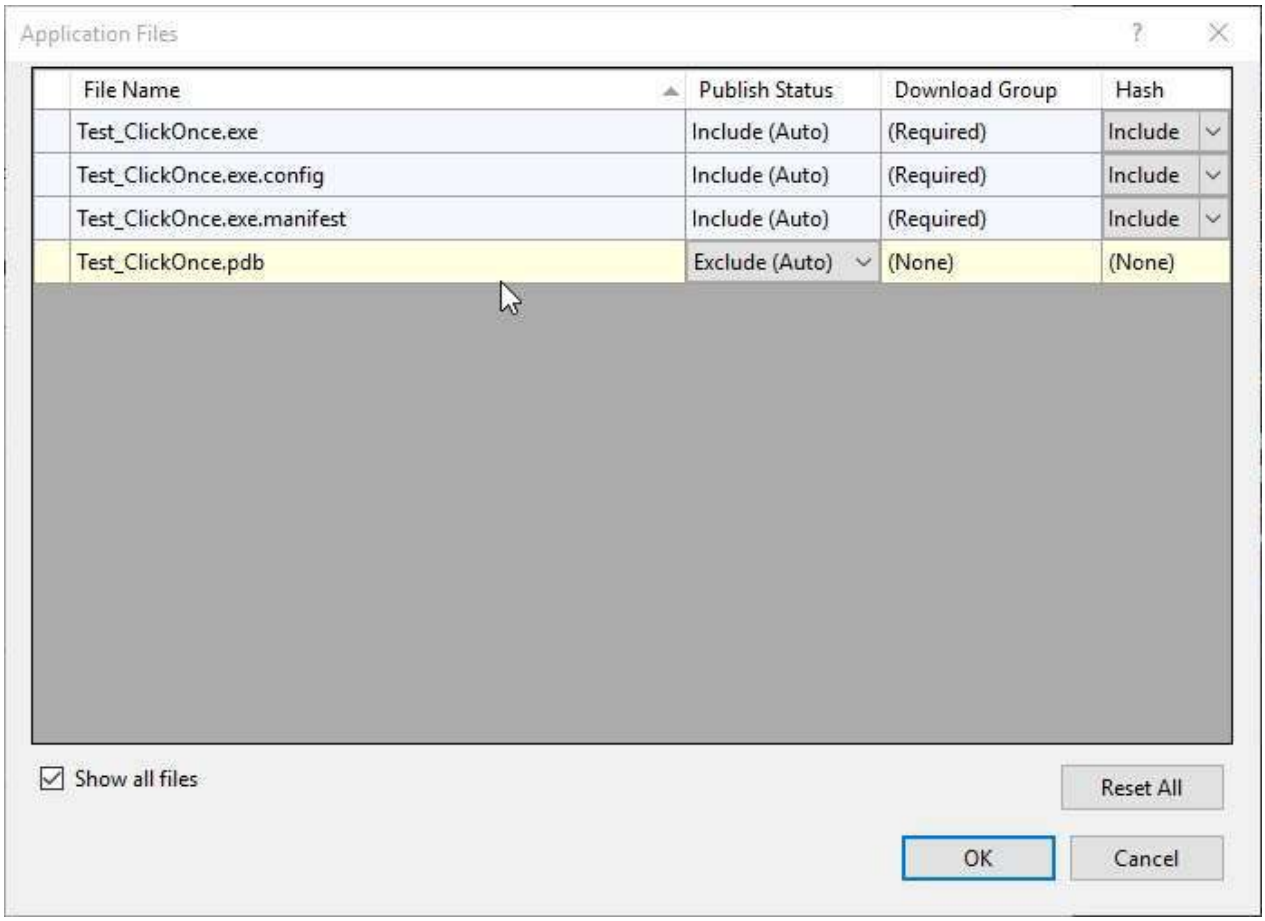
Une application ClickOnce ne peut pas être publiée sans être signée. La signature d'une application lui permet de déterminer la validité des mises à jour disponibles (si elles ont bien été publiées par la même personne que l'application initiale).
Par défaut, ces certificats sont auto-signés. L'application ClickOnce sait donc si une mise à jour est valide. Mais elle ne sait pas si celui qui a publié la première version est une personne de confiance.
C'est à cela que servent les certificats signés. Ils sont payants et permettent à l'utilisateur de savoir qui est réellement l'éditeur d'une application, le gérant d'un site, le propriétaire d'une adresse mail...

Pour ajouter votre certificat acheté, allez dans le menu *Signing* des paramètres de votre projet. Vous verrez les informations de votre certificat auto-signé, générées automatiquement. Les boutons *Select from Store* et *Select from File* vous permettent respectivement de récupérer un certificat déjà référencé ou un certificat se situant dans vos fichiers.



# Gestion des fichiers d'application

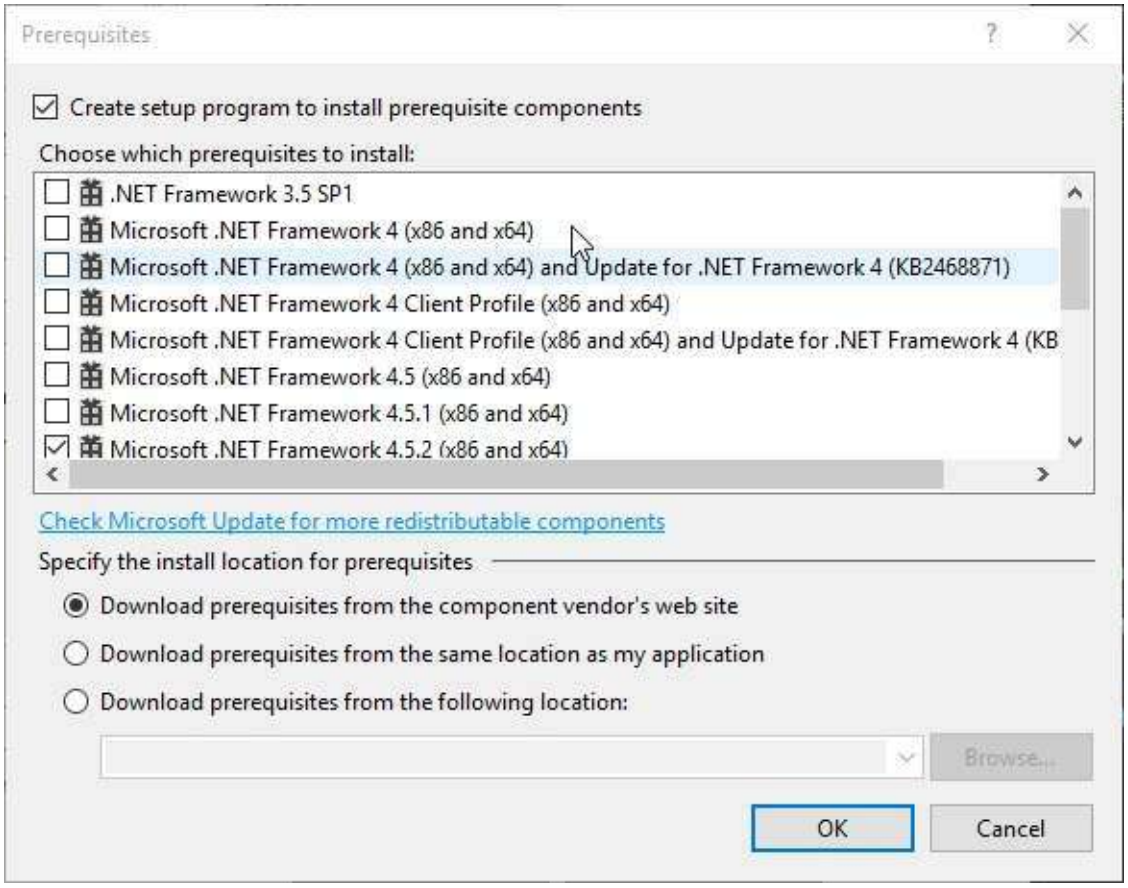
Dans le menu *Publish* des paramètres de votre application, une boite de dialogue vous permet de choisir les fichiers à inclure ou à exclure de votre application ClickOnce. Pour l'ouvrir, appuyez sur le bouton « Application Files ».




Les trois premiers fichiers sont nécessaires à notre application et il est donc impossible de les exclure.
Par contre, en activant « Show all files » on remarque un nouveau fichier : le fichier PDB (base de données du programme) de notre logiciel. Il est inutile dans notre cas : c'est un fichier contenant des informations pour le débogage. Il est donc désactivé par défaut.
Si vous avez d'autres fichiers, vous pourrez choisir comment les traiter : par exemple en les excluant, ou en les marquant comme étant des « fichiers de données ».

# Gestion des pré-requis

Vous pouvez également définir des pré-requis qui seront installés automatiquement par le fichier `setup.exe` . Ce fichier d'installation est généré automatiquement par Visual Studio et présent dans le répertoire où les informations du déploiement sont enregistrées.
Pour définir ces dépendances, cliquez sur le bouton *Prerequisites* du menu *Publish*. Vous pouvez également choisir à partir de quel site télécharger ces pré-requis : je vous conseille de laisser l'option par défaut, à savoir le centre de téléchargement Microsoft.



# Options de publication

Ces options, accessible grâce au bouton « Options », sont constitués de quatre onglets.

- L'onglet « description » permet de définir des informations sur le produit, qui apparaitront entre autres sur la page « Programmes et fonctionnalités » du panneau de configuration Windows. C'est ces informations qui seront enregistrées dans le registre.
- Dans l'onglet « deployment » on peut définir des paramètres liés au déploiement de l'application : faut-il générer la page web `publish.html` ? sous quel nom ? faut-il ajouter un `Autorun.inf` pour les déploiements sur CD
- « manifests » permet de définir certains paramètres liés au manifeste : faut-il autoriser les téléchargements en « un clic » ? faut-il créer un raccourci lors de l'installation ?
- Enfin, dans le menu « File Associations », on choisit des associations de fichier : un programme par défaut et une icône pour un certain format de fichier (par exemple, ouvrir des images Bitmap avec Paint).
- Ces associations sont créées lors de l'installation du logiciel par l'application ClickOnce.

# Création de son propre installeur

Si toutes les méthodes de personnalisation vues précédemment ne vous suffisent pas, nous allons voir ici comment créer votre propre installeur. Cela vous permet, entre autres, de créer une UI personnalisée pour votre installeur, d'ajouter des évènements lors de l'installation de l'application...

L'utilisateur pourra, grâce à celui-ci, télécharger **n'importe quelle** application *ClickOnce* en entrant l'URL de son manifeste dans une boîte de texte.
Nous rajouterons quelques instructions pour personnaliser notre installation, par exemple en affichant des boîtes de dialogue à un certain moment.

# Design de l'application

Avant tout, il nous faut *designer* notre installeur personnalisé. On peut utiliser Windows Forms, WPF ou même une application console. Ici, nous utiliserons trois contrôles Windows Forms :

- Une *TextBox* où l'on écrit l'URL du manifeste ClickOnce ;
- Une *RichTextBox* affichant les informations sur l'application ;
- Un *Label* affichant les informations sur le statut de l'installation ;
- Et enfin, une *ProgressBar* affichant la progression de l'installation.

# Création de notre classe installeur personnalisée

Afin de créer notre propre installeur, je recommande de mettre toute la logique liée à l'installation dans une classe à part. Cela simplifiera grandement votre code, surtout si vous comptez faire un installeur plus élaboré que le nôtre.

Créez donc un fichier `CustomClickOnceInstaller.cs` dans lequel vous insérerez ce code :

```
using System;
using System.Deployment.Application;
using System.Windows.Forms;

namespace VotreNamespace
{
    /// <summary>
    /// Installeur personnalisé
    /// </summary>
    public class CustomClickOnceInstaller
    {
        InPlaceHostingManager iphm;
    }
}
```

Vous remarquerez que nous avons importé trois *namespaces* :

- `System` ;
- `System.Deployement.Application`, permettant de créer un comportement personnalisé pour notre application ClickOnce ;
- `Windows.Forms` puisque nous intéragirons avec des contrôles WinForms.

L'`InPlaceHostingManager` nous permettra de télécharger/mettre à jour notre déploiement.

Il nous faut maintenant écrire une méthode installant notre application. Je recommande de faire passer une URL de manifeste dans les paramètres de la méthode afin de rendre votre installeur réutilisable pour d'autres logiciels.

```
/// <summary>
/// Installe une applicaiton ClickOnce.
/// </summary>
/// <param name="manifestUrl">URL du manifeste ClickOnce.</param>
public void Install(string manifestUrl)
{
    // :: Blocs try-catch
    // Permet de gérer les différents types d'erreur qui peuvent
    // être engendrés.
    try
    {
        // Création d'une classe Uri à partir de l'URL du manifeste
        var manifestUri = new Uri(manifestUrl);

        // :: Initialisation du InPlaceHostingManager
        // Le deuxième paramètre sert à spécifier si l'application ClickOnce est
        // téléchargée depuis un hôte (comme un navigateur web).
        iphm = new InPlaceHostingManager(manifestUri, false);
    }
    catch (UriFormatException)
    {
        MessageBox.Show("Erreur : L'URL du manifeste spécifiée est invalide.");
        return; // On arrête la fonction
    }
    catch (ArgumentException)
    {
        MessageBox.Show("Erreur : L'URL du manifeste spécifiée est invalide.");
        return; // On arrête la fonction
    }
    catch (PlatformNotSupportedException)
    {
        MessageBox.Show("Erreur : Votre système d'exploitation est trop ancien pour in
staller cette application.");
        return; // On arrête la fonction
    }

    // :: Ajout d'un évènement
    // La méthode iphm_GetManifestCompleted() se déclenchera lorsque
    // le manifeste de déploiement sera récupéré.
    iphm.GetManifestCompleted += iphm_GetManifestCompleted;

    // Récupère le manifeste sans bloquer le thread principal,
    // garantissant une meilleure expérience utilisateur.
    iphm.GetManifestAsync();
}
```

Lorsque le manifeste sera téléchargé, cette méthode sera exécutée :

```
/// <summary>
/// A lieu lorsque le manifeste de l'application ClickOnce est téléchargé.
/// </summary>
void iphm_GetManifestCompleted(object sender, GetManifestCompletedEventArgs e)
{
    // Vérification des erreurs
    if (e.Error != null)
    {
        MessageBox.Show("Erreur lors du téléchargement du manifeste : " + e.Error.Mess
age);
        return; // Annulation de l'installation.
    }

    // Vérifie si les privilèges requis sont satisfaits.
    try
    {
        // Si des élévations de privilèges sont requises, elles auront lieu.
        iphm.AssertApplicationRequirements(true);
    }
    catch (Exception)
    {
        MessageBox.Show("Erreur lors de la vérification des privilèges.");
        return;
    }

    // Mise à jour du titre de la fenêtre Main
    ((Main)Application.OpenForms[0]).Text = String.Format("Installation de '{0} [v{1}]
'", e.ProductName, e.Version);

    // Vérifie si l'application est déjà installée.
    if (e.ActivationContext.Form == ActivationContext.ContextForm.StoreBounded)
    {
        ((Main)Application.OpenForms[0]).statusLabel.Text = "Application déjà installé
e.";
return;
    }

    // Enregistrement des évènements :
    // - Changement de la progression du téléchargement ;
    // - Téléchargement terminé.
    iphm.DownloadProgressChanged += iphm_DownloadProgressChanged;
    iphm.DownloadApplicationCompleted += iphm_DownloadApplicationCompleted;

    // Téléchargement de l'application de manière asynchrone.
    iphm.DownloadApplicationAsync();
}
```

Ajoutons les deux derniers évènements nécessaires au fonctionnement de l'installeur :

```
/// <summary>
/// A lieu lorsque l'application est installée.
/// </summary>
void iphm_DownloadApplicationCompleted(object sender, DownloadApplicationCompletedEven
tArgs e)
{
    // Vérifier les erreurs
    if (e.Error != null)
    {
        MessageBox.Show("Erreur lors de l'installation de l'application : " + e.Error.
Message);
        return; // Annulation de l'installation.
    }

    // Dire à l'utilisateur que l'application est installée
    // puis fermer le logiciel.
    MessageBox.Show("Application installée.")
    Application.Exit(0)
}


/// <summary>
/// A lieu lorsque la progression du téléchargement de l'application change.
/// </summary>
void iphm_DownloadProgressChanged(object sender, DownloadProgressChangedEventArgs e)
{
    // Mettre à jour le pourcentage de l'installation.
    ((Main)Application.OpenForms[0]).progressBar.Value = e.ProgressPercentage;
}
```

# Bonus : exécuter certaines actions en fonction de la progression

Notre classe est bien belle, mais comment la rendre meilleure que l'installeur ClickOnce par défaut ?

Nous pouvons par exemple exécuter certaines actions lors Reprenons notre méthode `iphm_DownloadProgressChanged` :

```
/// <summary>
/// A lieu lorsque la progression du téléchargement de l'application change.
/// </summary>
void iphm_DownloadProgressChanged(object sender, DownloadProgressChangedEventArgs e)
{
    // Mettre à jour le pourcentage de l'installation.
    ((Main)Application.OpenForms[0]).progressBar.Value = e.ProgressPercentage;

    // Exécuter certaines actions en fonction du pourcentage
    switch(e.progressPercentage) {
        // Si un quart (25%) de l'application est installée, en informer l'utilisateur
        case 25:
            MessageBox.Show("Un quart de l'application installé !")
            break;

        // etc
        case 50:
            MessageBox.Show("La moitié de l'application installée !")
            break;

        case 75:
            MessageBox.Show("Les trois quarts de l'application installés !")
            break;
    }
}
```

Vous n'êtes pas obligé d'ouvrir des boîtes de dialogue, vous pouvez par exemple afficher des informations sur l'installeur pour le faire patienter...

# Lier la classe installeur à notre fenêtre

Pour ce faire, rien de plus simple, créez des évènements liés à chaque contrôleur et appelez les méthodes de notre classe. Par exemple, exécutez ce code lorsque le bouton est pressé :

```
CustomClickOnceInstaller installer = new CustomClickOnceInstaller();
installer.Install(textBox.Text);
```

# Conclusion

Voilà tout, vous êtes maintenant paré à utiliser ce merveilleux outil de déploiement 💪
Merci beaucoup d'avoir lu ce tutoriel. Si vous avez quelques questions à poser, n'hésitez pas à le faire dans les commentaires, je m'efforcerai d'y répondre le plus rapidement possible. :simple_smile: