Consommer les services Microsoft Translator dans une Application Windows WPF en C#

Article par Eric Vernié Microsoft France Division Plate-forme & Ecosystème

SOMMAIRE

Introduction

Développer des logiciels, n'a jamais été plus excitant qu'aujourd'hui. Nous sommes dans un monde de plus en plus connecté, interactif et mobile. Les ordinateurs n'ont jamais bénéficiés d'autant de puissance, permettant le développement d'interfaces graphiques innovantes, naturelles et plus intuitives, ouvrant la voie ainsi à de nouveaux scénarios et opportunités.

Un de ses scénarios est la possibilité de rajouter de la traduction et de la synthèse vocale à son application.
Dans cet article, je me propose de vous montrer, comment il est possible d'enrichir une application Riche Windows WPF, (par opposition à des applications Internet Web) en utilisant justement des services sur le Web de traduction et de synthèse vocale via les Microsoft Translator

L'architecture de notre application sera constituée de deux projets:

- Un projet **Traducteur**, de type bibliothèque de classe qui implémente les API Microsoft Translator, et qui masquera au client, l'implémentation aux appels du service Microsoft Translator.
- Un projet de type WPF **ClientTraducteur**, le client qui consommera notre bibliothèque **Traducteur**

Prérequis

Pour réaliser les exemples de cet article il vous faut :

- Visual Studio C# Express
- Créer une clé Microsoft Translator

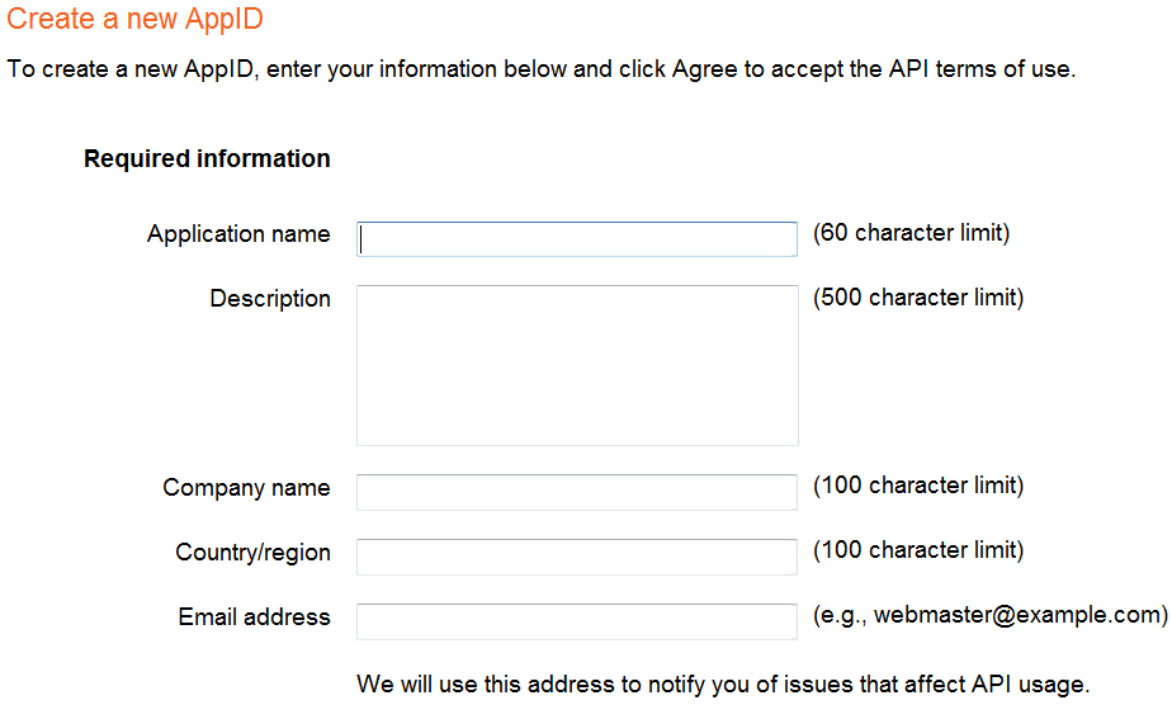
Microsoft Translator

Microsoft Translator est un outil qui permet de traduire et de synthétiser à la volée du texte dans différent langages.
Plusieurs interfaces sont disponibles, comme AJAX, HTTP, ou SOAP. Dans notre exemple, nous utiliserons l'interface SOAP, qui est la plus naturelle pour une application Windows WPF développée avec Visual Studio 2010 C# Express, si vous souhaitez en savoir plus n'hésitez pas à aller à l'adresse http://msdn.microsoft.com/en-us/library/ff512435.aspx

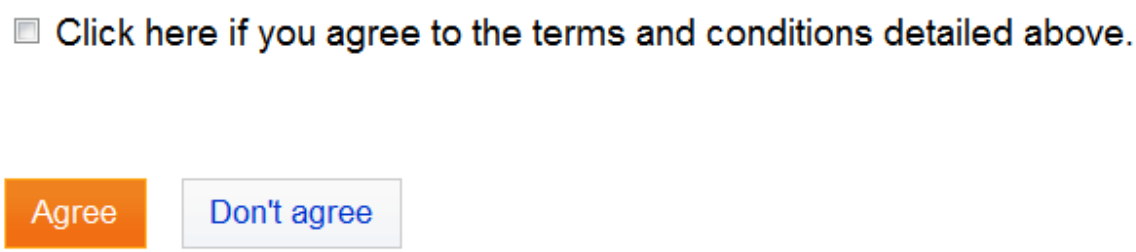
Création de la clé Microsoft Translator

1. Pour manipuler Microsoft Translator, il faut obtenir une clé (gratuite) à cette adresse http://www.bing.com/developers/createapp.aspx que nous réutiliserons avec les API Microsoft Translator par la suite.

Création d'une clé Microsoft Translator



2. Remplissez les champs avec vos informations
3. Cochez la case **Click here if you agree to the terms and conditions detailed above**



4. Appuyez sur le bouton **Agree**
5. Copiez ensuite la clé créée, pour la réutiliser ultérieurement

Création de la librairie d'appel à Microsoft Translator.

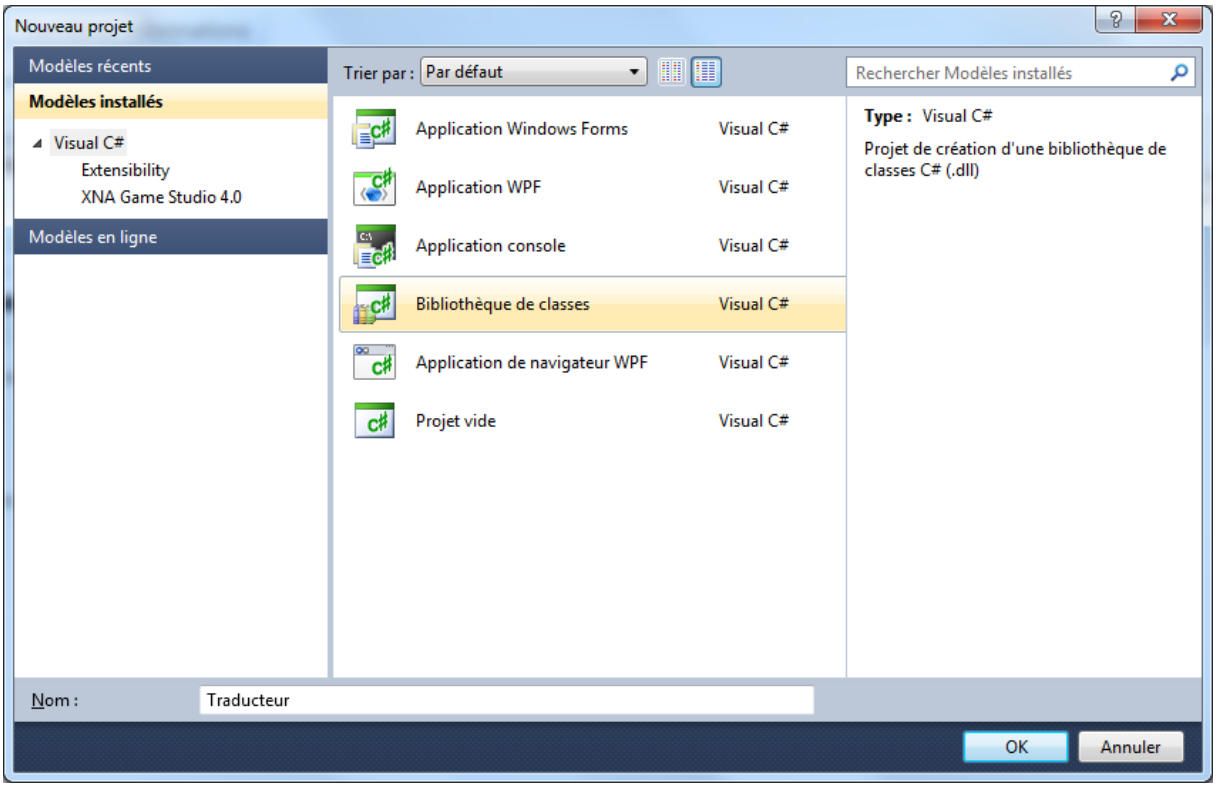
Maintenant nous allons créer un nouveau projet de type bibliothèque de classe, nommé **Traducteur**, qui fera le pont entre les interfaces SOAP de Microsoft Translator et notre client WPF. Nous pourrions bien évidement éviter cette étape, et faire appel directement dans notre client à Microsoft Translator. Mais en règle générale, il est toujours préférable de découper son application en couche, afin d'éviter qu'il y est trop d'adhérence entre les différents niveaux d'une application et arriver à des applications monolithiques, difficiles à maintenir. C'est également utile si vous souhaitez la réutiliser dans un autre projet.

Dans cette bibliothèque de classe, nous implémenterons les méthodes **Traduire** et **Synthetiser** qui ne feront rien de particulier, si ce n'est que d'appeler les méthodes correspondantes du service Microsoft Translator. Nous notifierons le client par l'intermédiaire d'un événement

RequeteTerminee, que la demande de traduction ou de synthèse a abouti, échouée, ou a été annulée.

Création de la classe APITranslator

1. Lancez Visual Studio C# Express
2. Choisissez le modèle Bibliothèque de classes et nommé le projet **Traducteur**

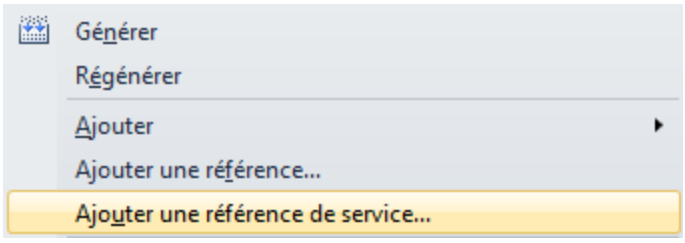


3. Renommez la classe **Class1** générée automatiquement par **APITranslator**

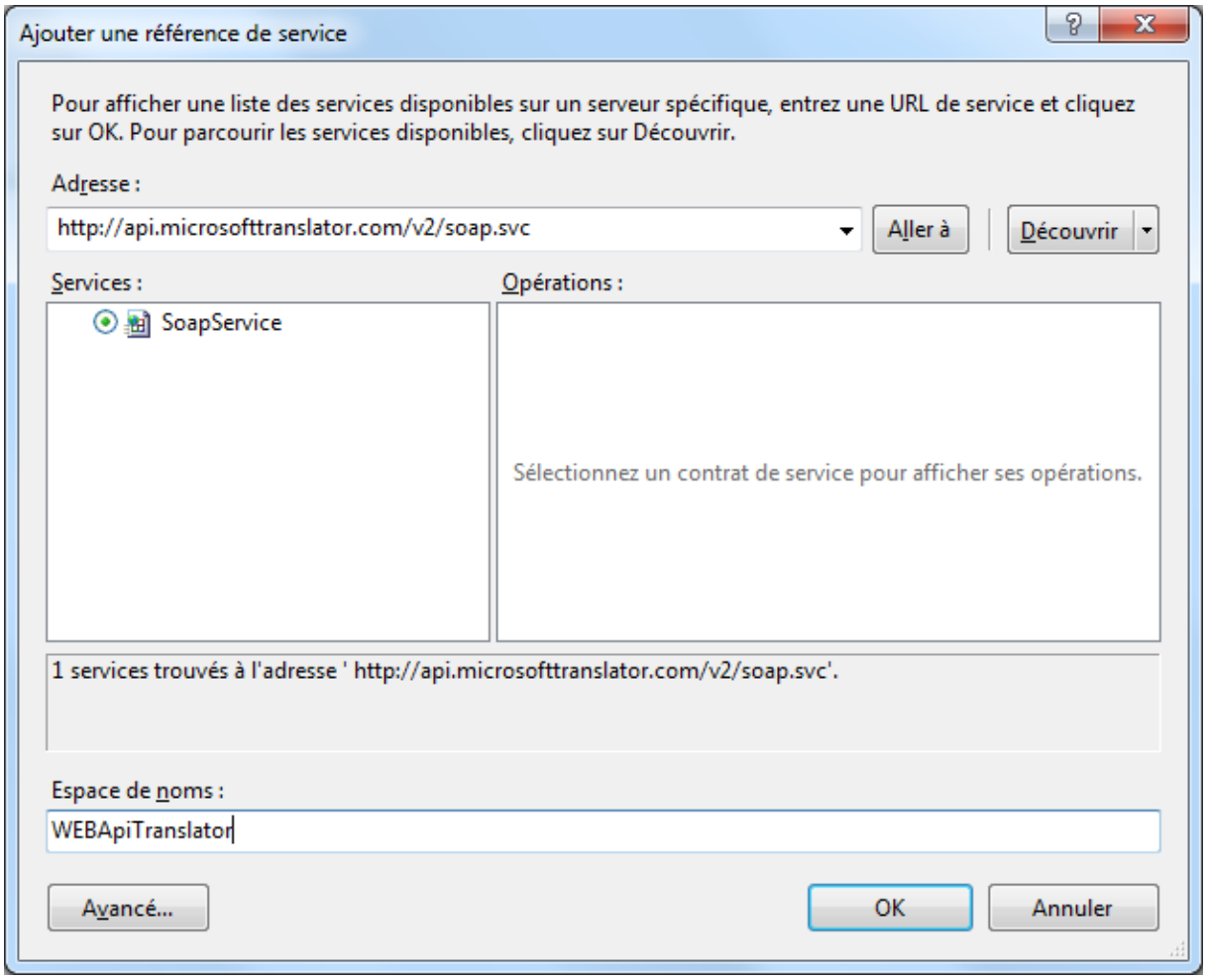
Création du Proxy WCF pour Microsoft Translator.

Pour pouvoir manipuler les API Microsoft Translator, il faut créer un proxy coté client, pour ce faire il vous faut utiliser un point d'entrée SOAP qui permettra à Visual Studio de créer toute la mécanique, en un mot tout le code Windows Communication Foundation que nous manipulerons par la suite.
Le point d'entrée est http://api.microsofttranslator.com/v2/soap.svc et sera toujours le même.
Pour en savoir plus sur les concepts WCF : Windows Communication Foundation

1. Dans le projet **Traducteur**, dans l'explorateur de solution, cliquez sur le bouton droit de la souris, et sélectionnez la commande **Ajouter une référence de service…** : comme illustré sur la figure suivante,

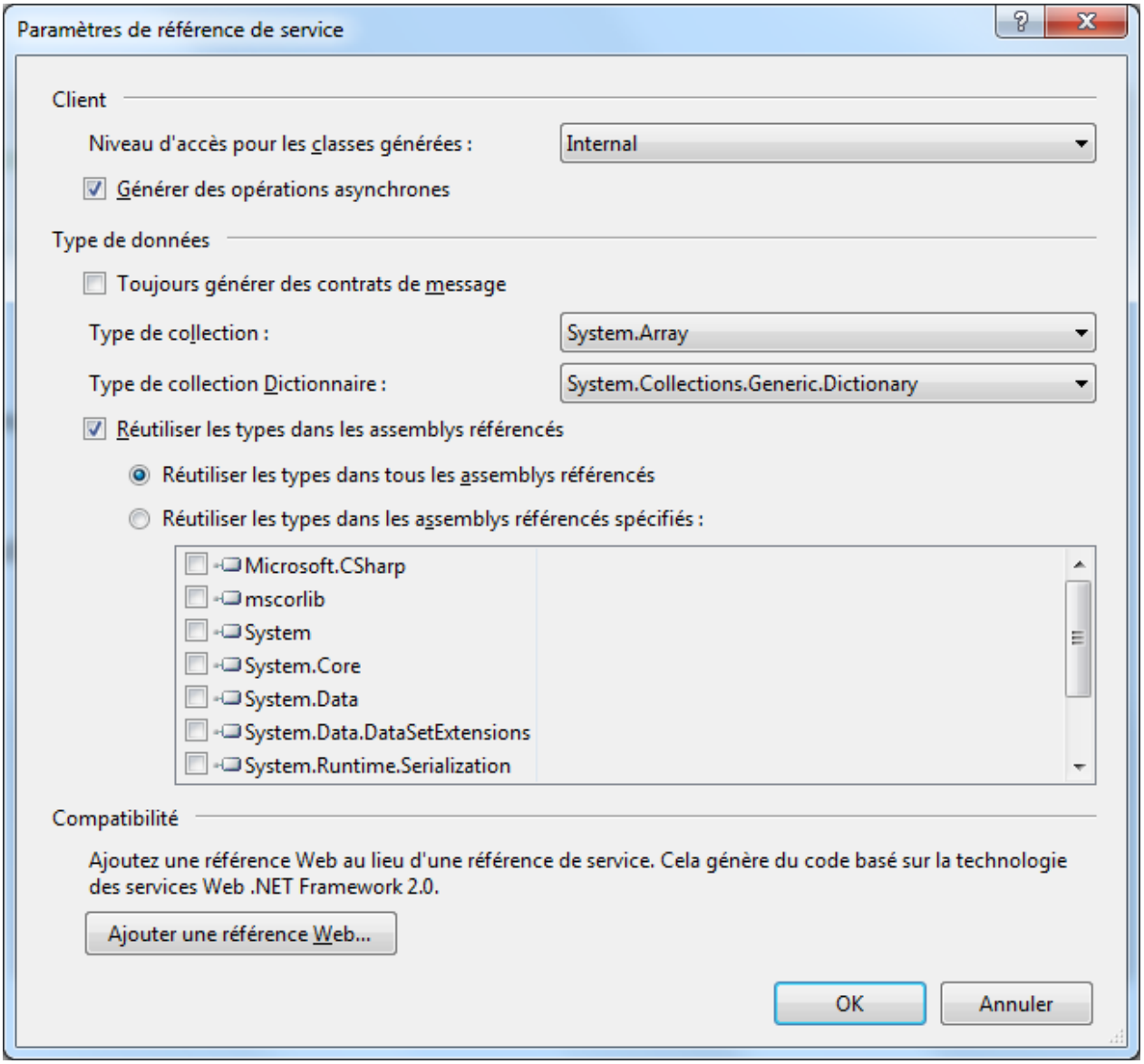


2. Dans le champ **Adresse**, rentrez l'adresse http://api.microsofttranslator.com/v2/soap.svc puis activez le bouton **Allez à**, comme illustré sur la figure suivante :



3. Donnez comme espace de nom **WEBApiTranslator.**
4. Sélectionnez le bouton **Avancé**

5. Cochez la case **Générer des opérations asynchrones**.



Remarque : En cochant cette case, nous allons créer toutes les APIs qui permettront d'exécuter les requêtes sur le Web en mode **asynchrone**, évitant de figer ainsi l'application lors d'une demande de traduction ou de synthèse vocale. Pour ceux qui seraient plus familier avec les Service Web non WCF, cliquez sur le bouton **Ajouter une référence Web**

6. Appuyez successivement sur les boutons **OK**, pour valider la création du proxy WCF.
   Remarque : Visual Studio vient d'ajouter de manière transparente toute la mécanique d'accès au service sous l'espace de noms **Traducteur.WEBApiTranslator**, que nous allons manipuler par la suite.

Implémentation des API Microsoft Translator

Dans la suite de notre démonstration, nous allons donc implémenter toute la mécanique de connexion au service via l'espace de noms **Traducteur.WEBApiTranslator**

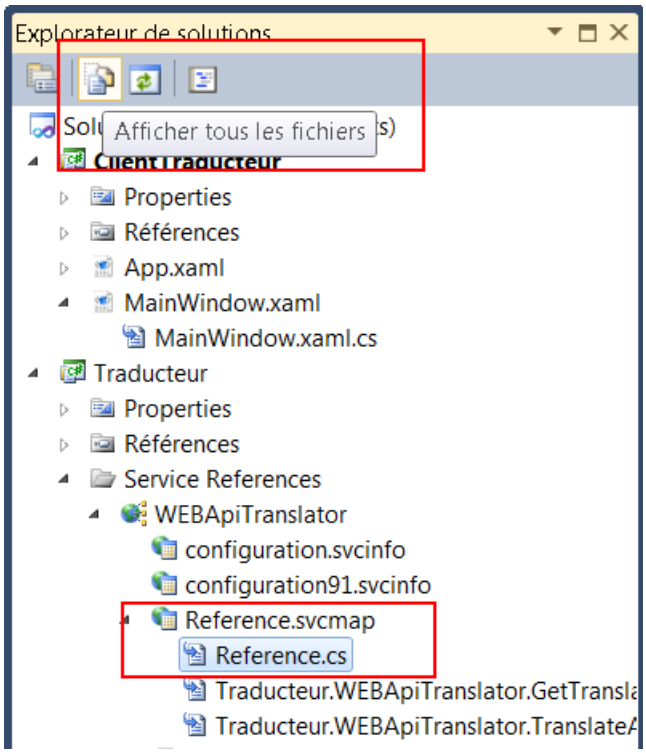
1. Dans l'entête du fichier ajoutez les importations d'espaces de noms suivantes :

```
using System.ServiceModel;
using Traducteur.WEBApiTranslator;
```

2. Dans la classe **APITranslator**, ajoutez le code suivant :

```
const String APPID = <VOTRE CLE ICI> ;
private LanguageServiceClient _clientTranslator;
```

Remarque : La classe **LanguageServiceClient**, est une des classes générées automatiquement par Visual Studio lors de l'étape **Création du Proxy Microsoft Translator**. Si vous souhaitez voir toutes les classes du proxy, activez le bouton **voir tous les fichiers** dans l'explorateur de solution, comme illustré sur la figure suivante :



3. Dans le constructeur de la classe **APITranslator** ajoutez le code suivant :

```
public APITranslator()
{
  BasicHttpBinding basicBinding = new BasicHttpBinding(BasicHttpSecurityMode.None);
  EndpointAddress endPoint = new
EndpointAddress("http://api.microsofttranslator.com/v2/soap.svc");
 _clientTranslator = new LanguageServiceClient(basicBinding,endPoint);
            _clientTranslator.TranslateCompleted += new
EventHandler<TranslateCompletedEventArgs>(_clientTranslator_TranslateCompleted);
_clientTranslator.SpeakCompleted += new
EventHandler<SpeakCompletedEventArgs>(_clientTranslator_SpeakCompleted);
        }
```

Je ne rentre pas dans le détail de l'implémentation de l'appel WCF, vous retrouverez toute l'aide à cette adresse Programmation WCF de base. Ici nous utiliserons comme couche transport, le protocole http de base sans sécurité particulière, et nous définissons notre point de terminaison http://api.microsofttranslator.com/v2/soap.svc.

Ensuite nous nous abonnons aux évènements **TranslateCompleted** et **SpeakCompted** afin d'être notifié qu'un résultat est arrivé ou pas. Rappelez-vous, nous allons exécuter des demandes asynchrones, afin de ne pas figer l'interface graphique de notre client. Le faite de s'abonner à ces évènements, nous permettra très simplement de réagir uniquement lorsqu'une réponse est arrivée.
Notez également que les méthodes de rappel seront **_clientTranslator_TranslateCompleted et _clientTranslator_SpeakCompleted**, que nous détaillerons, un peu plus loin.

4. Implémentons les méthodes **Traduire** et **Synthetiser**. Pour ce faire ajoutez le code suivant :

```
public void Traduire(String texte, String de, String vers)
{
      _clientTranslator.TranslateAsync(APPID, texte, de, vers);
}

public void Synthetiser(String texte, String langage)
{
        _clientTranslator.SpeakAsync(APPID ,texte,langage,"audio/wav");
}
```

Comme vous pouvez le constater, ces méthodes sont des plus simple, elles ne font que transmettre la demande aux méthodes **TranslateAsync** et **SpeackAsync**. La première prend comme paramètre, la clé d'identification que vous avez obtenue, le texte à traduire, le langage du texte à traduire, et le langage destination, et la seconde, la clé, le texte à synthétiser et le langage destination. Pour cette dernière, Microsoft Translator renverra un flux audio au format wav.

Maintenant nous allons implémenter toute la mécanique qui permettra de notifier notre client qu'un résultat est arrivé

5. La première chose à faire est de créer son propre évènement, pour ce faire procédez comme suit :
   a. Créez les arguments de l'évènement, en ajoutant le code suivant :

```
public class TraducteurEventArgs :EventArgs
    {
        public String Resultat;
        public Boolean EstSynthetiser;
        public TraducteurEventArgs(String r, Boolean s)
        {
            Resultat = r;
            EstSynthetiseur = s;
        }
    }
```

   TraducteurEventArgs, permettra de renvoyer au client, le résultat de la requête, ainsi qu'une variable booléenne (**EstSynthetiser**) pour identifier si la notification est une traduction ou une demande de synthèse vocale.
   b. Pour créer un évènement, il faut déclarer un délégué,

```
public delegate void TraducteurHandler(Object sender,TraducteurEventArgs e);
```

   Qui prend comme paramètre notre argument de type **TraducteurEventArgs**
   Notez que nous calquons la signature de notre évènement au modèle .NET préconisé, mais ce n'est pas une obligation, seulement une bonne manière de faire.

   c. Ensuite déclarez l'évènement lui-même de type **TraducteurHandler**.

```
public event TraducteurHandler RequeteTerminee;
```

   d. Enfin, nous implémentons la méthode **SurRequeteTerminee** qui lèvera l'évènement afin de notifier le client

```
protected void SurRequeteTerminee(TraducteurEventArgs e)
        {
            if (RequeteTerminee != null)
            {
                RequeteTerminee(this, e);
```

```
            }
        }
```

6. Pour finir, il suffit d'implémenter les deux méthodes de rappels **_clientTranslator_TranslateCompleted** et **_clientTranslator_SpeakCompleted,** afin que notre librairie soit notifiée elle-même qu'un résultat est arrivé, et de notifier par la suite le client de ce résultat.

```
void _clientTranslator_TranslateCompleted(object sender, TranslateCompletedEventArgs e)
        {
            //…code omis pour plus de clarté
            TraducteurEventArgs args = new TraducteurEventArgs(e.Resultat, false);
            RequeteTerminee(this, args);
}
void _clientTranslator_SpeakCompleted(object sender, SpeakCompletedEventArgs e)
        {
            //…code omis pour plus de clarté
            TraducteurEventArgs args = new TraducteurEventArgs(e.Resultat, true);
            RequeteTerminee(this, args);
}
```

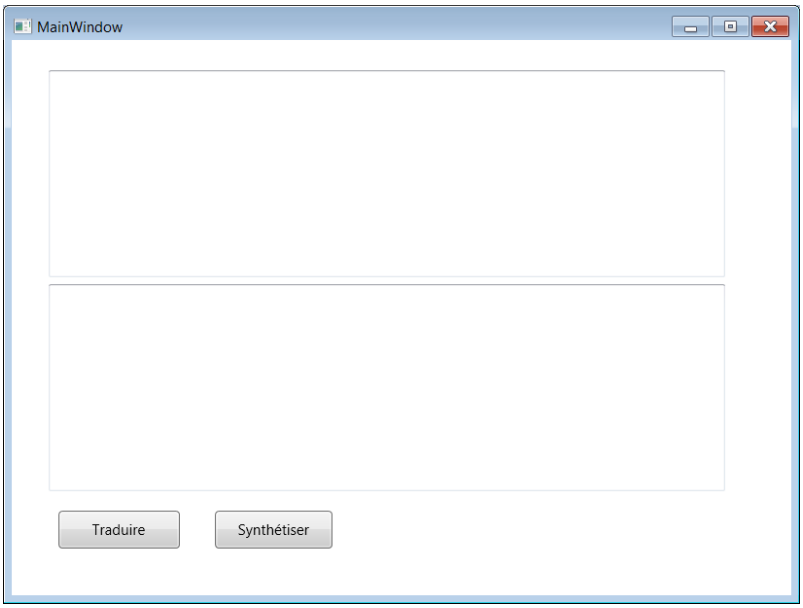
Notez que le code des deux méthodes sont très proche l'une de l'autre, à l'exception du faite que nous passons le paramètre **EstSynthetiser** à **false** ou à **true**, en fonction de la demande Traduire ou Synthétiser.

Notez également que nous avons omis du code pour plus de clarté dans cet article, mais qu'il est possible de tester avec le paramètre **e.cancelled**, si la requête a été annulée ou de tester si une erreur est survenue, à l'aide du paramètre **e.error.**

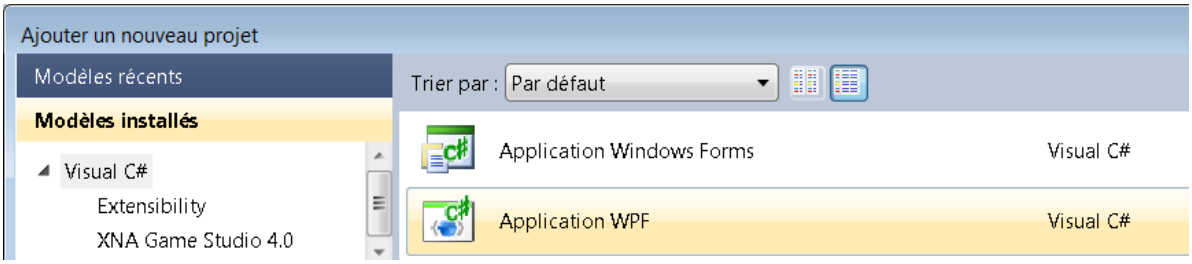
Voilà notre bibliothèque est désormais prête à l'emploie, nous allons maintenant créer le client qui la consommera.

Création du client WPF

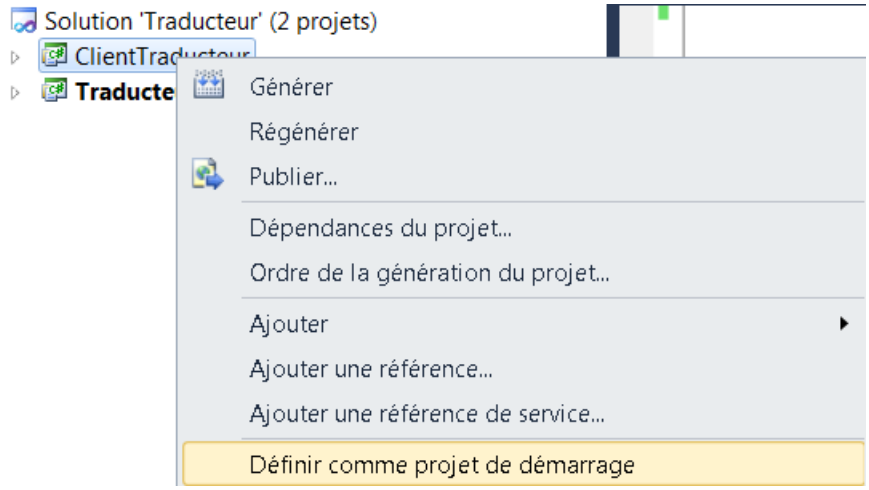
Nous allons créer un client très simple, qui aura pour interface graphique, deux contrôles de type **TextBox**, l'un pour le texte à traduire, l'autre pour le texte traduit. Ce dernier servira également comme texte de synthèse vocale, et deux boutons l'un pour la traduction, l'autre pour la synthèse vocale, comme illustré sur la figure suivante :



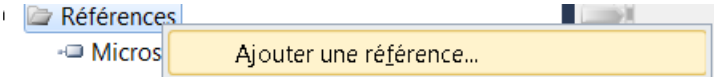
1. Ajoutez un nouveau projet de type WPF à la solution existante

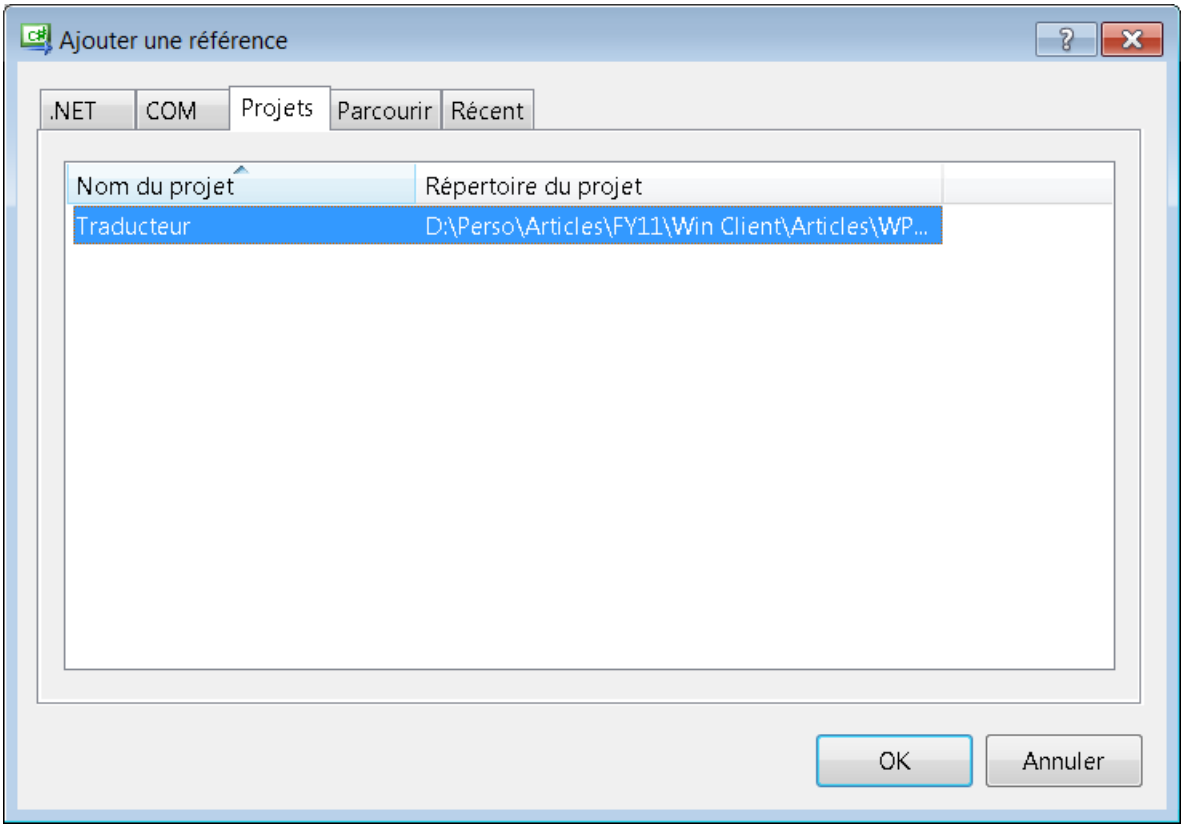


2. Appelez-le **ClientTraducteur**.
3. Définissez ce nouveau projet en tant que projet de démarrage



4. Double-cliquez sur le fichier **MainWindow.xaml**, afin d'ouvrir le designer XAML, et ajoutez lui deux contrôles TextBox, que vous nommez respectivement **txtATraduire** et **txtTraduit**.
5. Puis ajoutez deux boutons que vous nommez **cmdTraduire** et **cmdSynthetiser**
6. Ajoutez comme référence, le projet **Traducteur**. Afin de pouvoir manipuler notre librairie que nous venons de créer. Cliquez sur le bouton droit de la souris au niveau du nœud **référence**, et sélectionnez la commande **Ajouter une référence**





7. Ouvrez le fichier **MainWindow.xaml.cs**, afin d'accéder au code sous-jacent de l'interface graphique
8. Ajoutez-y le code suivant :

```
using Traducteur ;
using System.Media;
```

L'importation de l'espace de nom **System.Media** nous permettra de manipuler la classe **SoundPlayer()**. L'espace de nom **Traducteur**, est là pour manipuler notre librairie.

9. Déclarez en globale à la classe **MainWindows** les deux variables suivantes :

```
Traducteur.APITranslator _traducteur=null;
SoundPlayer _snd=null;
```

10. Dans le constructeur de la classe **MainWindows**, ajoutez le code suivant :

```
_traducteur = new APITranslator();
_traducteur.RequeteTerminee += new
APITranslator.TraducteurHandler(_traducteur_RequeteTerminee);
_snd = new SoundPlayer();
```

Vous remarquerez que nous abonnons notre client à l'évènement **RequeteTerminee**, qui prend comme méthode de rappel **_traducteur_RequeteTerminee** que nous détaillons à l'étape suivante.

11. Ajoutez une méthode **_traducteur_RequeteTerminee**

```
void _traducteur_RequeteTerminee(object sender, Traducteur.TraducteurEventArgs e)
        {
            if (e.EstSynthetiseur == false)
            {
                txtTraduit.Text = e.Resultat;
            }
            else
            {
```

```
                _snd.SoundLocation = e.Resultat;
                _snd.PlaySync();
            }
        }
```

L'essentiel du code de notre client réside ici. Si la variable **e.EstSynthetiser** est positionnée à false, alors c'est une traduction demandée, sinon c'est de la synthèse vocale. Pour ce dernier cas, nous utilisons la classe **SoundPlayer()**, pour jouer un son.

12. Maintenant sur l'évènement click des deux boutons, vous allez ajouter respectivement le code suivant :

    Pour le bouton **cmdTraduire**

```
private void cmdTraduire_Click(object sender, RoutedEventArgs e)
        {
            _traducteur.Traduire(txtAtraduire.Text , "fr", "en");
        }
```

    Pour le bouton **cmdSynthetiser**

```
        private void cmdSynthetiser_Click(object sender, RoutedEventArgs e)
        {
            _traducteur.Synthetiser(txtTraduit.Text, "en");
        }
```

    Vous noterez ici que nous utilisons par défaut, une traduction du Français vers l'Anglais, et une synthèse vocale en Anglais. Mais sachez que vous pouvez obtenir par code les différents langages supportés pour l'une ou l'autre des opérations. En effet Microsoft Translator fournit à ce sujet les méthodes **GetLanguagesForTranslate /GetLanguagesForTranslateAsync**, et **GetLanguagesForSpeak/ GetLanguagesForSpeakAsync** , que vous pouvez utiliser afin d'éviter de coder en dur ces deux paramètres.
13. Compilez l'application, exécutez-la et à vous de jouer.

Conclusion :

Dans cet article, nous n'avons fait qu'aborder une petite partie de Microsoft Translator, mais il existe pléthore d'API Microsoft Translator qui vous permettrons d'enrichir votre application Windows.
Par exemple :

- Il est possible de détecter automatiquement le langage en fonction d'un texte,
- De récupérer toutes les translations possibles en fonction d'un Texte. Différent de la méthode Translate que nous avons utilisé, car elle ne renvoie qu'un seule résultat.
- Et d'autres, que je vous laisse découvrir http://msdn.microsoft.com/en-us/library/ff512435.aspx

Dans un prochain article nous verrons comment implémenter les services Bing de recherche, et les consommer dans une application Windows.