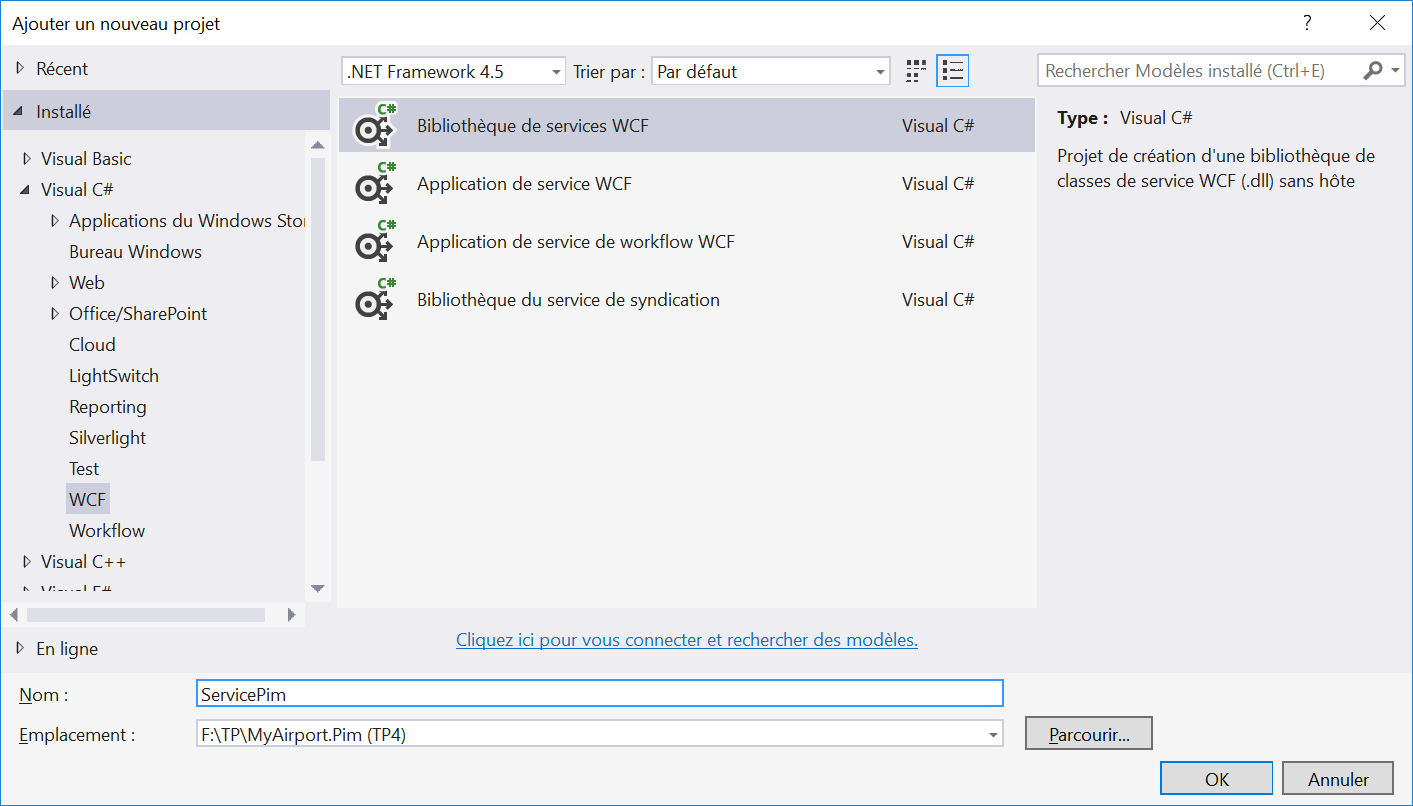
# TP4. Implémentation du service.

## Objectifs :

* Implémentation du service WCF.
* Connection du client au service WCF.
* Créer une solution d’hébergement du service.

## Implémentation du service WCF:

Ajouter un nouveau projet WCF de type bibliothèque de service et le nommer ‘ServicePim’.

Attention WCF propose deux types de projets de service, les bibliothèques de services WCF et les applications de services WCF, ce deux types de projets sont très proches et sont utilisés pour la création de services WCF, les bibliothèques de services ont pour vocations à être hébergées dans des applications indépendante (Services Windows, application Windows, console,…) alors que les applications de service ont vocation à être déployées dans IIS et sont donc vues plutôt comme des pages Web.

Renommer le fichier IService1.cs en IServicePim.cs via le menu contextuel Renommer.

Supprimer la classe CompositeType du fichier IService1.

Dans l’interface IServicePim supprimer le contenu et ajouter les définitions des méthodes :

namespace MyAirport.Pim.Service

{

[ServiceContract]

public interface IServicePim

{

[OperationContract]

BagageDefinition GetBagageById(int idBagage);

[OperationContract]

BagageDefinition GetBagageByCodeIata(string codeIata);

[OperationContract]

int CreateBagage(BagageDefinition bag);

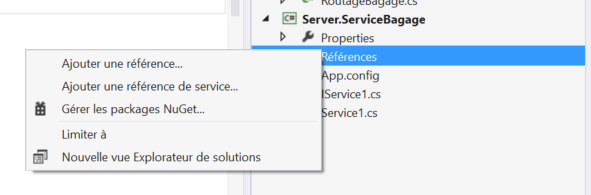
}

}

La définition de cette interface est très proche de la définition de la classe abstraite Models.AbstractDefinition car dans notre cas, l’ensembles des méthodes des ‘models’ sont exposées.

Vous pouvez constater que Visual Studio ne reconnait les classes définies dans le namespace ‘MyAirport.Pim’.

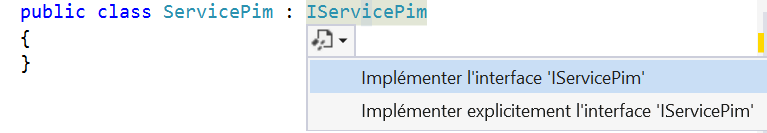
Dans le projet ServicePim ajouter une référence aux projets Entities, Models.AbstractDefinition et Models.Factory.



Les attributs OperationContract sont utilisés par WCF pour connaitre les méthodes qui devront être exposées par le service. L’attribut ServiceContract permet à WCF de savoir que cette interface expose des méthodes de service. L’ensemble des classes nécessaires à WCF sont regroupées dans l’assembly System.ServiceModel.

Renommer le fichier Servcie1 en ServicePim c’est ici que sera implémenté le code du service.

Supprimer le contenu de la classe ServicePim et utilisé le smartTag pour génèrer les prototypes des méthodes de IServicePim.



Notre service expose maintenant des méthodes qui exploitent des objets BagageDefinition, il convient donc d’indiquer à WCF comment Sérialiser ces objets.

Ajouter l’attribut DataContract a la classe BagageDefinition.

Ajouter l’attribut DataMember aux propriétés de cette classe.

Les attributs DataContract et DataMember sont définis dans la librairie System.Runtime.Serialization.

Il convient maintenant d’implémenter le comportement de notre service dans la classe ServicePim.

Remplacer le code de la méthode GetBagageById par le code ci-dessous :

public class ServicePim : IServicePim

{

public BagageDefinition GetBagageById(int idBagage)

{

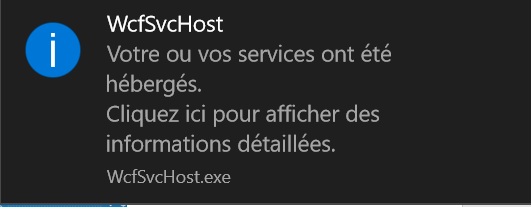
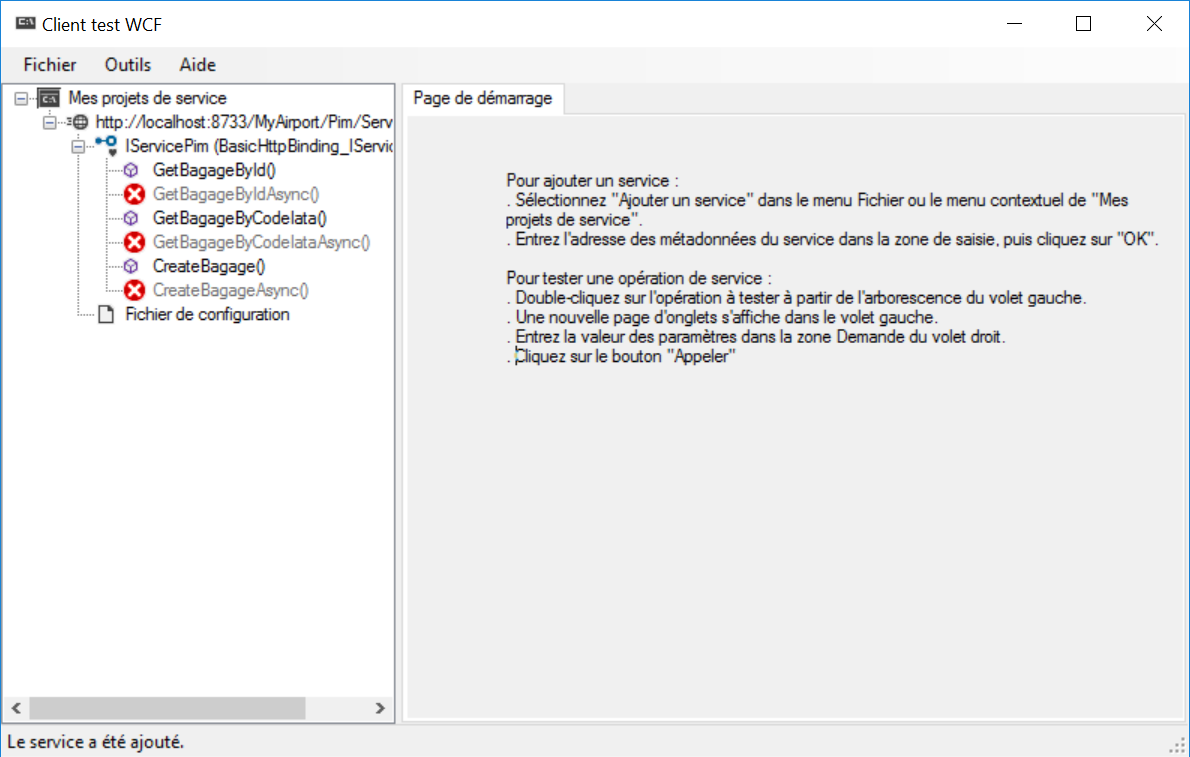
return MyAirport.Pim.Models.Factory.Model.GetBagage(idBagage);

}

Implémenter les autres méthodes de la classe. Attention ici la méthode GetBagageByCodeIata ne retourne pas une liste de BagageDefinition mais un unique objet BagageDefinition, dans un premier temps retourner uniquement le premier bagage de la liste s’il y en a un.

Définir le projet ServicePim comme projet de démarrage et compiler le projet.

Deux fenêtres doivent s’ouvrir après la compilation l’une nommée ‘Hôte de service WCF’ et l’autre ‘WCFSvcHost’ ces fenêtres correspondent respectivement au client et au serveur de notre service web. Elles n’ont pas vocation à perdurer mais nous permettent de tester notre service rapidement.



Si votre service ne démarre pas correctement vérifier :

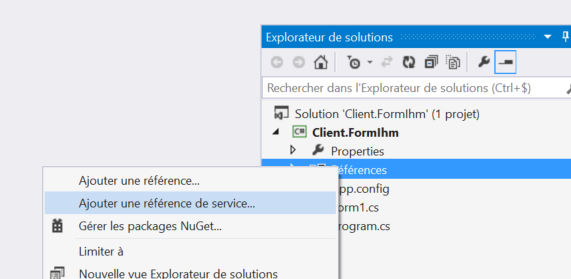
Dans le fichier App. Config que le nom du service et son namespace soit correcte ainsi que celui de l’interface de service.

Si l’ouverture du service est en échec vérifier que Visual Studio est bien démarré avec des droits administrateurs nécessaires pour l’ouverture d’un port sur le réseau.

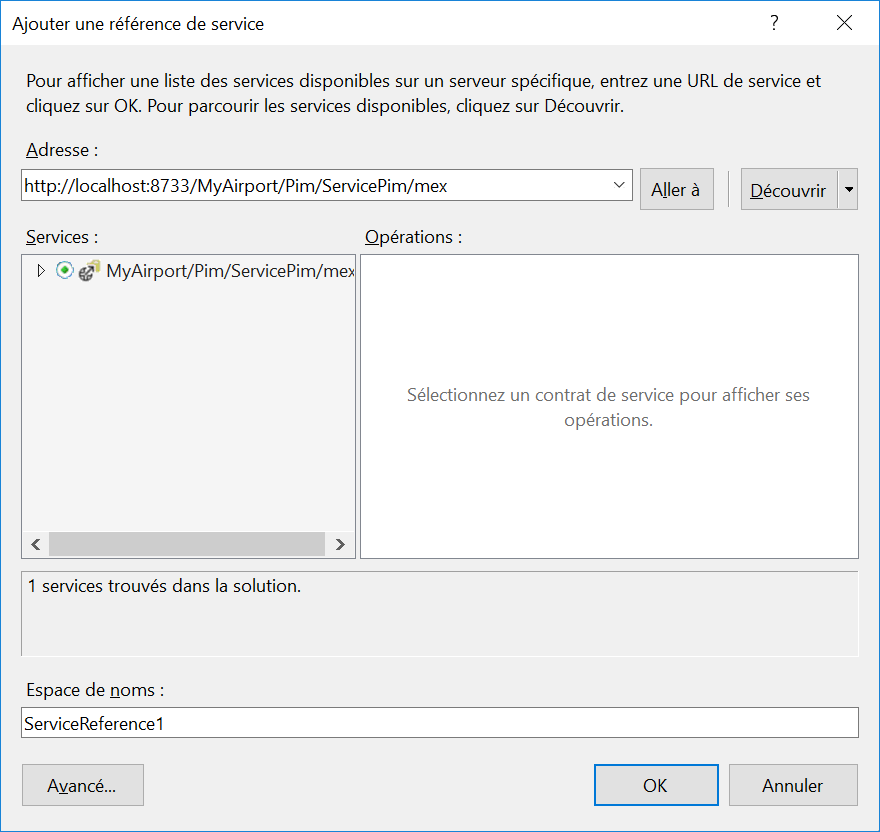
Dans la fenêtre ‘Hôte de service WCF’, Le service retourne null pour la recherche d’un bagage bien que le codeIata saisie soit en base de données. Cela est dû au fait que notre fichier app.config ne contient plus de définition pour la Factory et c’est donc le modèle Natif qui est chargé par la Factory.

## Connection du client au service WCF.

Une fois le service testé et valider retourner dans le projet client.FormIhm et ajouter une référence de service



Et cliquer sur découvrir, votre service devrait apparaitre dans la liste des sélections. Modifier l’espace de nom en ServiceReferencePim et valider sur OK.



Supprimer maintenant toutes les références aux assembly MyAirport, elles ne seront plus nécessaires car le client utilise dorénavant le service WCF.

ServiceBagageReferencePim.ServicePimClient proxy = null;

public Form1()

{

InitializeComponent();

proxy = new ServiceBagageReferencePim.ServicePimClient();

}

Supprimer les éléments Factory et connexionString du fichier app.config du client.

Dans la classe Client.FormIhm.Pim compléter la méthode buttonFind\_Click permettant la recherche d’un bagage via son codeIata de la manière suivante :

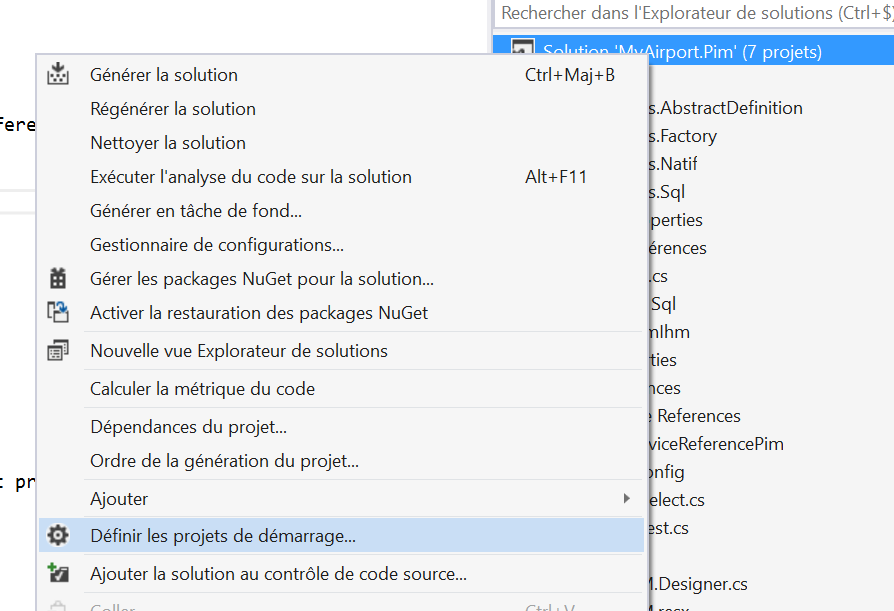
private void buttonFind\_Click(object sender, EventArgs e)

{

Client.FormIhm.ServiceReferencePim.BagageDefinition bag = proxy.GetBagageByCodeIata(this.tbCodeIata.Text);

}

Dans la solution définir les projets de démarrages et sélectionner démarrage de plusieurs projets Client.FormIhm et ServicePim.



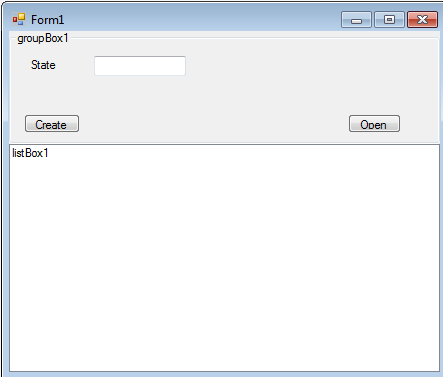
### Hébergement du service.

Dans le cas du projet MyAirport, le service sera base sur une solution auto hébergée à titre purement pédagogique. Cette solution permet de disposer d’un meilleur contrôle sur le comportement du service afin d’analyser et de mieux comprendre les différents paramétrages possibles, de plus cette solution requière un minimum de configuration et de droits ce qui est pratique dans un environnement de test et permet de se focaliser uniquement sur des problématiques liées au service.

WCF implémente une classe ServiceHost offrant toutes les fonctionnalités nécessaires à l’hébergement d’un service WCF. La propriété State permet de connaitre l’état du service dont le cycle de vie est représenté ci-dessous.

Ajouter un nouveau projet Serveur.Host de type application Windows, à la solution MyAirport.Pim.Server.

Disposer l’interface afin de ressembler à la capture ci-dessous :



Dans la classe Form1, créer une nouvelle instance de ServiceHost en fournissant en paramètre le type du service à héberger.

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private ServiceHost host = null;

private void buttonCreate\_Click(object sender, EventArgs e)

{

host = new ServiceHost(typeof(MyAirport.Pim.Service.ServicePim));

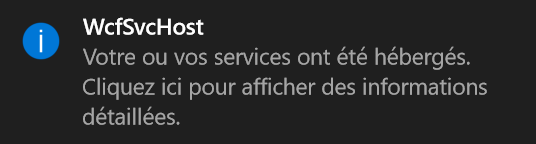
}

}

Ajouter les références aux librairies ServicePim et System.ServiceModel.

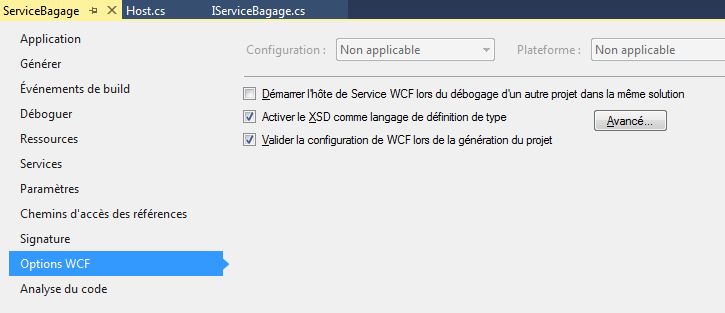
Copier le fichier app.config du projet ServicePim dans le fichier app.config du projet WinodwsFormsHost.

Nous avons jusqu’à présent tester le service sans qu’il ne soit nécessaire de créer une instance de la classe Host pour héberger notre service. Cela est rendu possible par VisualStudio qui va au démarrage du service exécuter une application qui va se charger d’héberger notre service. Le démarrage de cette application nous est signalé par l’apparition d’un popup.



Bien que cette fonctionnalité soit pratique pour tester notre service en cours de développement, il est nécessaire de désactiver cette fonctionnalité lorsque nous souhaitons créer notre propre implantation de la classe host, car dans le cas contraire les deux applications vont tenter d’écouter les requêtes clientes sur les mêmes points de terminaisons, ce qui provoquera un échec, car deux applications ne peuvent ouvrir simultanément un même socket. Il va donc falloir désactiver cette fonctionnalité dans les propriétés du projet.

Sélectionner le projet ServicePim, faite un clic droit et choisir propriétés, puis sélectionner l’onglet Option WCF et enfin décocher l’option ‘Démarrer l’hôte de service…’



Définir le projet WindowsFormsHost comme projet de démarrage, compiler et tester le bon fonctionnement du client.

Afin de visualiser le cycle de vie du service, il est nécessaire de s’abonner à l’ensemble des événements liés au changement d’état du host.

private void buttonCreate\_Click(object sender, EventArgs e)

{

host = new ServiceHost(typeof(MyAirport.Pim.Service.ServicePim));

host.Closed += host\_State;

host.Closing += host\_State;

host.Faulted += host\_State;

host.Opened += host\_State;

host.Opening += host\_State;

}

private void host\_State(object sender, EventArgs e)

{

this.textBox1.Text = this.host.State.ToString();

}

Le comportement de l’hôte de service peut être modifié via l’attribut InstanceContextMode, attribut placé en tête de la classe qui implémente le service.

[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]

Mettre en place la logique d’affichage des informations dans l’hôte de service.

private void button2\_Click(object sender, EventArgs e)

{

if(this.host != null)

if(this.host.State == CommunicationState.Opened)

{

this.host.Close();

this.button2.Text = "Ouvrir";

}

else

{

this.host.Open();

this.listBox1.Items.Clear();

foreach (var item in host.Description.Behaviors)

{

if (item is System.ServiceModel.ServiceBehaviorAttribute)

{

this.listBox1.Items.Add(((System.ServiceModel.ServiceBehaviorAttribute)item).InstanceContextMode.ToString());

}

}

foreach (var item in host.Description.Endpoints)

{

this.listBox1.Items.Add(item.Name);

}

this.button2.Text = "Fermer";

}

}

Testez le service avec 2 clients simultanés et modifier la valeur de InstanceContextMode afin de bien comprendre les différences entre ces différents modes.