Atelier

Utiliser les notifications de type Push

Version: 1.0.0

Contenu

Aperçu

L'expérience utilisateur est la plus importante des caractéristiques de Windows Phone 7. Des efforts supplémentaires ont été faits pour s'assurer que les applications ne consomment pas trop les ressources de la batterie. Du coup, Windows Phone 7 ne permet pas à votre application d'exécuter du code dans un processus d'arrière-plan. Cela signifie que votre application ne peut pas interroger régulièrement un Web Service pour obtenir une information. Le service de notification de Push de Microsoft (Microsoft Push Notification, MPN) compense cette restriction et vous permet d'envoyer des messages à un périphérique Windows Phone même si votre application n'est pas en cours d'exécution.

Il y a trois acteurs majeurs dans ce service :

- L'application Web de backoffice, ou **cloud service,** qui communique avec l'application Windows Phone 7
- Le **périphérique Windows Phone**—et non votre application WP7, car si votre application est en cours d'exécution, il y a peu de chance qu'elle utilise le service de notification. Au contraire, si votre application n'est pas en cours d'exécution, alors le client du service de notification joue un rôle majeur. Ce client est un service, tournant sur WP7, qui implémente le protocole « Client To Server ».
- Le **service de notification (**Microsoft Push Notification MPN) —qui envoie, de la part de votre application Web de backoffice, les messages sous forme de notification de type "push" aux périphériques WP. Ce service est conçu pour fournir aux services de backoffice un canal dédié et persistant dont le but est de permettre d'envoyer des notifications à un périphérique mobile.

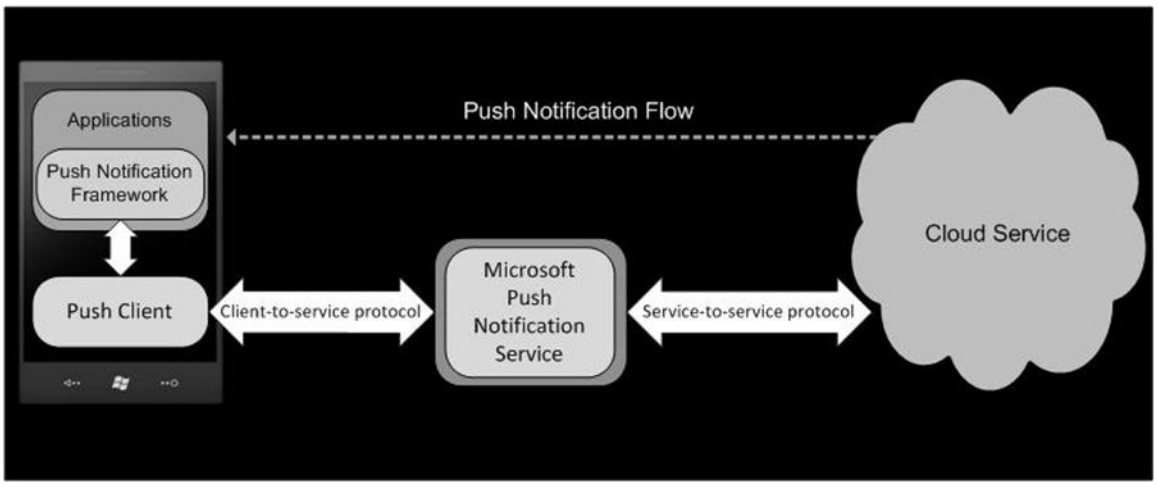


Figure 1

Notifications de type Push

Lorsqu'un service a besoin d'envoyer une notification de type Push à un périphérique, il envoie la demande de notification au service MPN, qui redirige la notification vers l'application ou vers le périphérique. Le tout s'effectue en fonction du type de notification qui est envoyé.

Le client du service MPN, sur le périphérique, reçoit la notification de type Push à travers un canal de notification. Lorsque le canal est créé, un abonnement l'est également afin de permettre au service de backoffice d'envoyer des notifications au canal. Un canal est représenté par une URI qui contient toutes les informations associées à l'enregistrement.

Lorsque l'application reçoit les notifications de type Push, elle peut accéder au service de backoffice au travers de son protocole de communication pour récupérer les informations dont elle a besoin.

Les applications d'aujourd'hui tendent à partager de multiples écrans qui seront vus par des ordinateurs installés localement, dans le « cloud » ou sur des téléphones. Le service MPN permet une intégration de ces 3 types de périphériques au travers d'une API simple et unifiée.

Cet atelier introduit les notifications de type Push et l'utilisation de services HTTP dans Silverlight. Vous allez découvrir comment créer la logique, côté serveur, nécessaire pour envoyer des messages au service MPN. Vous verrez également comment créer une simple application Windows Phone 7 qui servira de client pour recevoir les notifications. L'application cliente recevra des notifications de mise à jour de la météo. L'application métier, côté serveur, sera une simple application WPF qui enverra les alertes de météo aux différents clients connectés, grâce au service MPN. Lorsque l'application cliente Windows Phone 7 recevra l'alerte, elle affichera les informations reçues.

Objectifs

A la fin de cet atelier :

- Vous serez familier avec les possibilités de communication des applications Windows Phone 7
- Vous serez familier avec le concept de notification de type Push et les fonctionnalités du téléphone qui le permettent
- Vous comprendrez comment fonctionnent les notifications de type Push, sur le téléphone et dans le « cloud »
- Vous serez capable d'utiliser les services de notification (de type Push) du téléphone
- Vous serez capable d'utiliser des clients Web pour vous enregistrer auprès des notifications de type Push

- Vous serez capable d'utiliser les fonctionnalités d'interrogation du statut réseau pour afficher le statut de connectivité du téléphone
- Vous aurez créé une application Silverlight qui :
    - S'enregistre auprès du service de notification (de type Push)
    - Gère les évènements de réception de notification, à l'exécution
    - Affiche des notifications sur le Shell

Prérequis

Les éléments suivants sont nécessaires pour réaliser cet atelier :

- Microsoft Visual Studio 2010 Express for Windows Phone or Microsoft Visual Studio 2010
- Windows Phone Developer Tools

Installation

Pour faciliter cet atelier, beaucoup de code utilisé est disponible sous la forme de Code Snippet, dans Visual Studio. Pour installer ces Codes Snippets:

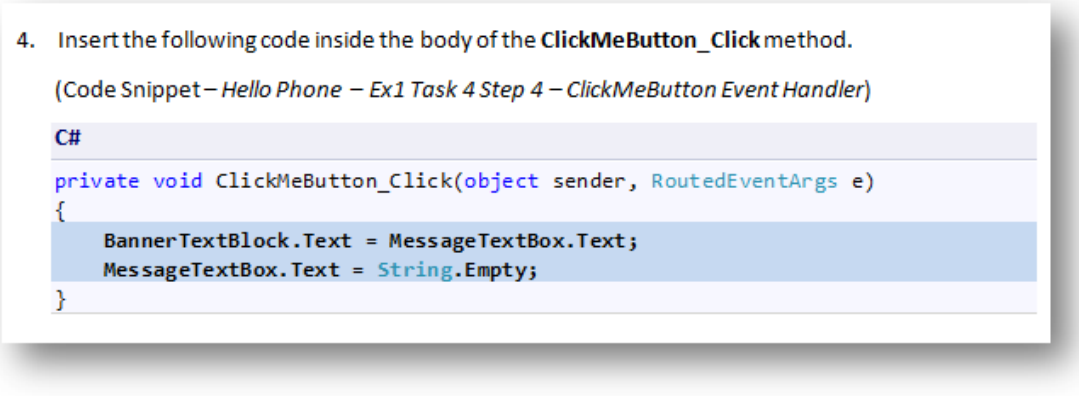
1. Exécutez le fichier .**vsi** situé dans le répertoire **Source\Setup**.

> **Note:** Si vous avez des problèmes pour exécuter l'installeur de "Code Snippets", vous pouvez les installer à la main en copiant tous les fichiers situés dans le répertoire **Source\Setup\CodeSnippets** vers le répertoire suivant :
>
> **\My Documents\Visual Studio 2010\Code Snippets\Visual C#\My Code Snippets**

Utiliser les « Code Snippets »

Avec les “Code Snippets”, vous avez tout le code nécessaire à portée de main. Ce document vous explique exactement comment les utiliser.

Note : Les noms de Code Snippet dans les images suivantes sont montrés à titre d’exemple.


4. Insert the following code inside the body of the **ClickMeButton_Click** method.

(Code Snippet – *Hello Phone – Ex1 Task 4 Step 4 – ClickMeButton Event Handler*)

C#

```
private void ClickMeButton_Click(object sender, RoutedEventArgs e)
{
    BannerTextBlock.Text = MessageTextBox.Text;
    MessageTextBox.Text = String.Empty;
}
```

Figure 2

Utilisation les Code Snippets de Visual Studio pour insérer du code dans votre projet

Pour ajouter un Code Snippet dans Visual Studio, vous devez simplement placer le curseur à l’endroit où vous désirez que le code soit inséré. Commencez ensuite à taper le nom du snippet (sans les espaces), puis sélectionnez, dans l’IntelliSense, celui qui vous intéresse. Pressez ensuite la touche Tab 2 fois : le code sera inséré après la position du curseur:

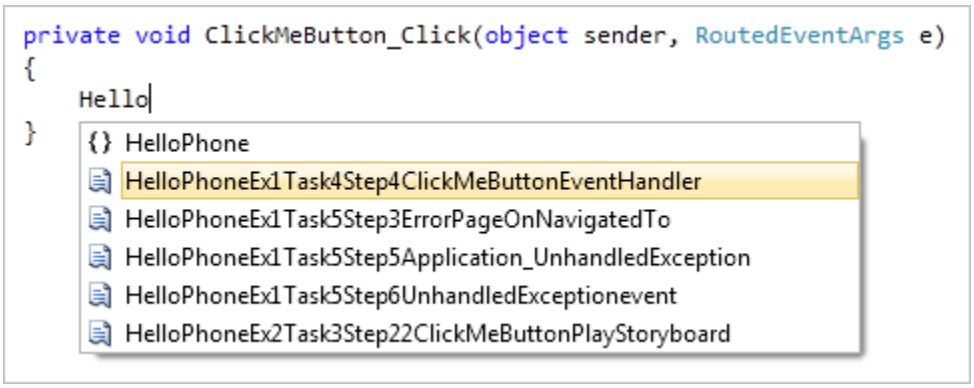


Figure 3

Saisie du nom du snippet

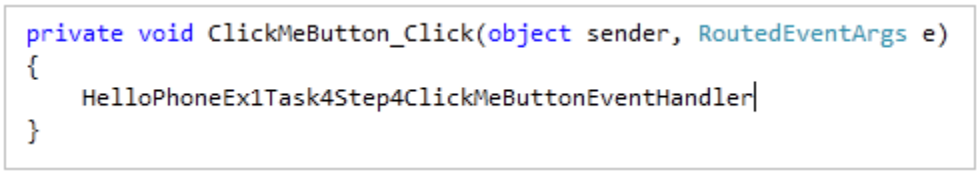


Figure 4

Appui sur la touche Tab pour sélectionner le snippet surligné

```
private void ClickMeButton_Click(object sender, RoutedEventArgs e)
{
    BannerTextBlock.Text = MessageTextBox.Text;
    MessageTextBox.Text = String.Empty;
}
```

Figure 5

Appuyez sur la touche Tab une nouvelle fois pour insérer le snippet

Pour insérer un snippet en utilisant la souris plutôt que le clavier, faîtes un clic droit dans le bloc de code où vous souhaitez insérer le snippet, sélectionnez **Insert Snippet** suivi par **My Code Snippets,** et enfin, choisissez le snippet qui vous intéresse dans la liste.

Pour en savoir plus sur les Code Snippets, rendez-vous ici: http://msdn.microsoft.com/en-us/library/ms165392.aspx.

Exercices

Cette partie comprend les exercices suivants :

1. Introduction aux notifications de Windows Phone pour les mises à jour
2. Introduction aux notifications pour les alertes

Durée estimée: **90 minutes**.

Exercice 1: Introduction aux notifications de mise à jour de Windows Phone 7

Dans cette section, vous allez ouvrir la solution de démarrage et :

- Implémenter la notification, côté serveur, et les services d'enregistrement
- Créer une application Windows Phone 7
- Créer un canal de notification et vous abonner aux évènements de ce canal
- Recevoir et gérer les évènements qui proviennent du service de notification de Push de Microsoft

Nous utiliserons Microsoft Visual Studio 2010 Express for Windows Phone pour développer notre application et l'émulateur Windows Phone pour la débugger. Durant l'atelier, nous ajouterons un élément de projet Silverlight spécifique à Windows Phone : la page portrait de Windows Phone.

Note : Les étapes dans le lab illustrent les procédures en utilisant Microsoft Visual Studio 2010 Express for Windows Phone, mais elles s'appliquent de la même façon si vous utilisez Microsoft Visual Studio 2010 accompagné du Windows Phone Developer Tools.

Etape 1 – Création du service de backoffice « Weather Service »

Dans cette tâche, vous utilisez une solution de démarrage pour Microsoft Visual Studio 2010 Express for Windows Phone ou Microsoft Visual Studio 2010. Cette solution inclut une application WPF qui est utilisée pour envoyer des messages aux applications Windows Phone 7 en utilisant le service MPN. Cette application WPF héberge également un service WCF REST, que nous créerons par la suite. Le projet a déjà tous les éléments de configuration nécessaires.

1. Ouvrez Microsoft Visual Studio 2010 Express for Windows Phone depuis **Démarrer | Tous les programmes | Microsoft Visual Studio 2010 Express**.

Note importante : Vous devez lancer Visual Phone 2010 Express for Windows Phone ou Microsoft Visual Studio 2010 en **mode Administrateur** pour que le service WCF hébergé dans l'application puisse fonctionner. Pour plus d'informations sur comment créer et héberger des services WCF hostés, référez-vous à l'article MSDN ci-contre : (http://msdn.microsoft.com/en-us/library/ms731758.aspx). Pour ouvrir Visual Studio 2010 Express for Windows Phone ou Visual Studio 2010 en mode Administrateur, utilisez le raccourci Microsoft Visual Studio 2010 Express for Windows Phone dans **Démarrer | Tous les programmes | Microsoft Visual Studio 2010 Express** ou le

raccourci Microsoft Visual Studio 2010 dans **Démarrer | Tous les programmes | Microsoft Visual Studio 2010**, faîtes un clic droit sur l'icône et sélectionnez « Exécuter en tant qu'Administrateur » dans le menu contextuel. La fenêtre de l'UAC sera alors lancée. Cliquer sur « Oui » pour permettre à Visual Studio 2010 Express for Windows Phone ou Visual Studio 2010 de s'exécuter avec des privilèges élevés.

2. Dans le menu **Fichier**, sélectionnez **Ouvrir un projet**.

Visual Studio 2010 : Dans le menu Fichier, pointez sur **Ouvrir** et ensuite, sélectionnez **Projet/Solution**.

3. Naviguez à l'endroit où se situent les fichiers de démarrage : *{LAB_PATH}\Source\Ex1-RawNotifications\Begin*, sélectionnez **PushNotifications.sln**, et cliquez sur Open

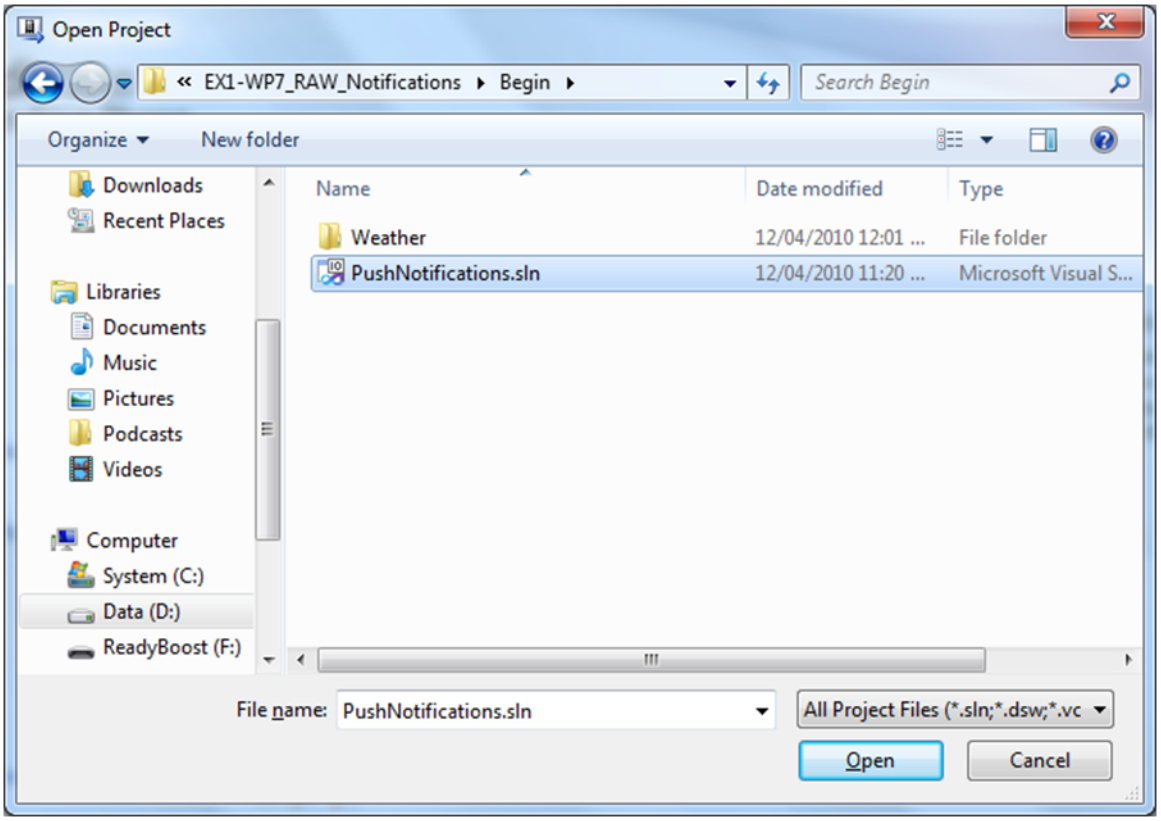


Figure 6

Ouverture du projet de démarrage

4. Examinez les projets ouverts:

   a. Vous disposez d'une application WPF standard.

Note: Cette application cible le Framework.NET 4 et non pas le Client Profile .NET 4 pour pouvoir supporter les services WCF RESTful auto-hébergés.

b. L'écran MainWindow de l'application WPF contient le contrôle utilisateur PushNotificationsLogViewer et le convertisseur StatusToBrush.

c. Le projet "Service" inclut l'interface de définition du serveur WCF RESTful qui sera implémenté.

d. En plus des assemblys traditionnelles, l'application référence également System.ServiceModel et System.ServiceModel.Web pour les services WCF RESTful que nous utiliserons pour permettre au téléphone d'enregistrer son URI.

5. Compilez et exécutez l'application. Familiarisez-vous avec puis fermez là et retournez dans Visual Studio. L'application WPF permet aux applications WP7 d'enregistrer l'URI qui recevra les notifications de la part du service MPN.

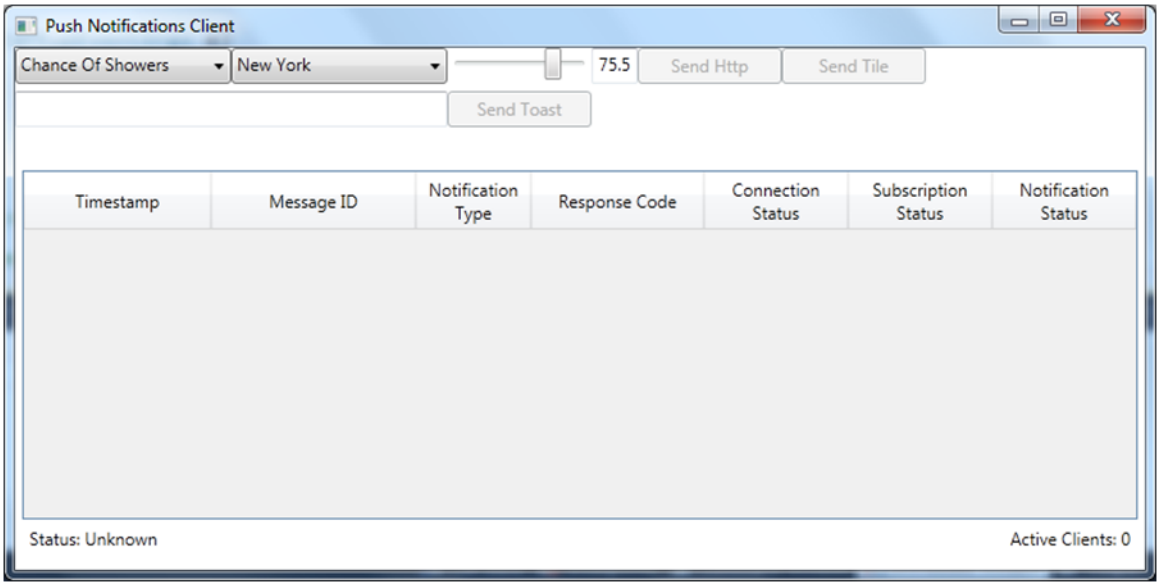


Figure 7

Application d'envoi de notifications de type Push

Durant les prochaines étapes, vous allez créer une instance d'un service WCF et l'initialiser. Pour communiquer avec le service MPN, l'appelant doit fournir l'URI du canal enregistré par le périphérique spécifique. Vous allez créer un tel canal et l'enregistrer dans la prochaine tâche de cet atelier.

6. Ajoutez une nouvelle classe au répertoire **Service** du projet: Faîtes un clic droit sur le nom du répertoire puis sélectionnez **Add** => **Class** :

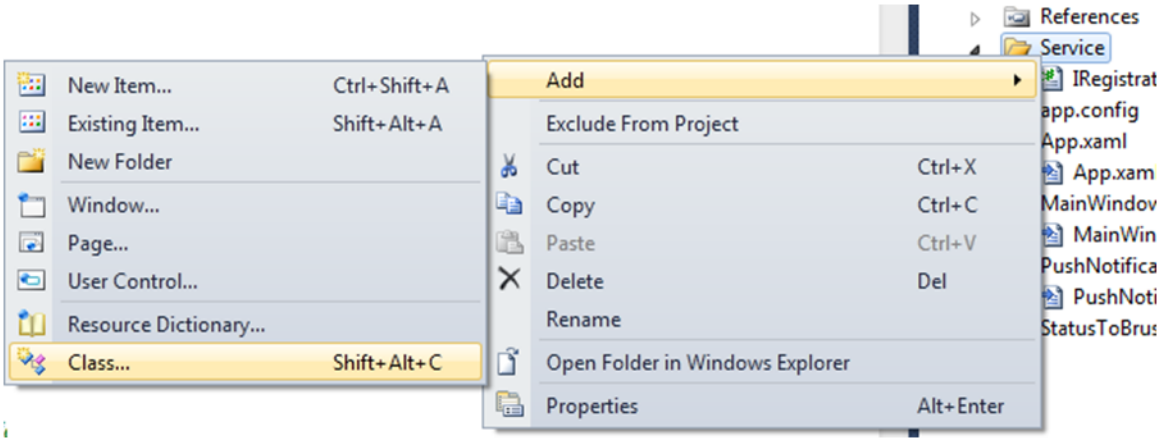


Figure 8

Ajout d'une nouvelle classe au projet

7. Nommez cette classe **RegistrationService** et cliquez sur Add.

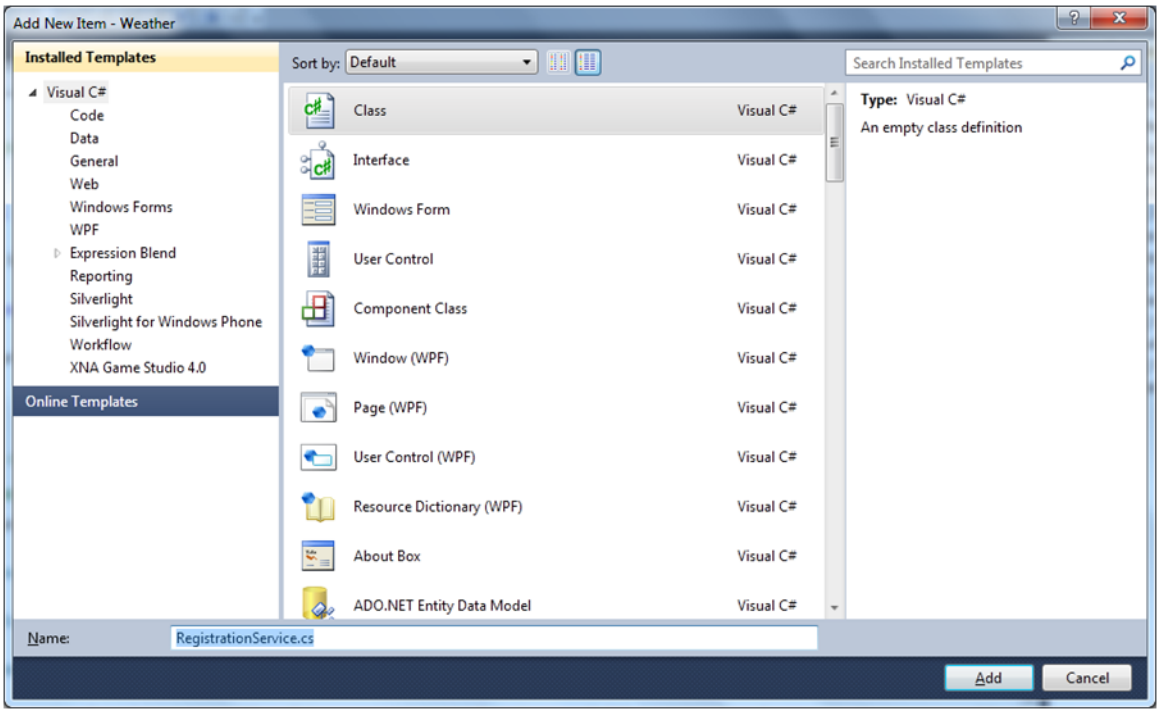


Figure 9

Nommage de la nouvelle classe

8. Ouvrez la classe nouvellement créée.
9. Marquez la class comme publique et faîtes-lui implémenter l'interface **IRegistrationService**.

(Code Snippet – *Using Push Notifications – Registration Service – Implementing the interface*)

C#

```
public class RegistrationService : IRegistrationService
{

}
```

10. Cliquez sur **IRegistrationService**, et ouvrez le menu contextuel en passant votre souris sur le signe "_".

Figure 10

Ouvrir les options de l'interface

11. Sélectionnez **Implement interface IRegistrationService** pour implémenter l'instance par défaut. Visual Studio 2010 créée les squelettes de propriétés ou méthodes pour vous.

Figure 11

Implémentation de l'interface

12. Le code de la classe doit donc ressembler à ceci :

C#

```
public class RegistrationService : IRegistrationService
{
    #region IRegistrationService Members
    public void Register(string uri)
    {
        throw new NotImplementedException();
    }

    public void Unregister(string uri)
    {
        throw new NotImplementedException();
    }
    #endregion
}
```

Le service d'enregistrement conserve la liste des URI qui lui sont rattachées. Lorsqu'un périphérique WP enregistre son URI (reçue du service MPN), nous devons informer le client WPF qu'il doit mettre à jour l'information correspondante dans l'interface graphique. Le service vérifie également que le client ne s'est pas déjà enregistré afin d'éviter les doublons.

13. Pour permettre cette fonctionnalité, ajoutez ces variables de classe :

(Code Snippet – *Using Push Notifications – Registration Service – Class variables*)

C#

```
public static event EventHandler<SubscriptionEventArgs> Subscribed;

private static List<Uri> subscribers = new List<Uri>();
private static object obj = new object();
```

14. Remplacez le corps de la fonction **Register** avec l'extrait de code suivant:

(Code Snippet – *Using Push Notifications – Registration Service – Register function body*)

C#

```
Uri channelUri = new Uri(uri, UriKind.Absolute);
Subscribe(channelUri);
```

15. Remplacez le corps de la fonction **Unregister** avec l'extrait de code suivant :

(Code Snippet – *Using Push Notifications – Registration Service – Unregister function body*)

C#

```
Uri channelUri = new Uri(uri, UriKind.Absolute);
Unsubscribe(channelUri);
```

16. Créez une nouvelle région avec les méthodes d'aide **Subscribe** et **Unsubscribe** comme le montre le code suivant. Ces fonctions ajoutent et suppriment une URI de la liste des URI :

(Code Snippet – *Using Push Notifications – Registration Service – Subscription and Unsubscription region*)

C#

```
#region Subscription/Unsubscription logic
private void Subscribe(Uri channelUri)
{
    lock (obj)
    {
            if (!subscribers.Exists((u) => u == channelUri))
            {
                subscribers.Add(channelUri);
            }
    }
    OnSubscribed(channelUri, true);
}

public static void Unsubscribe(Uri channelUri)
{
    lock (obj)
```

```
    {
        subscribers.Remove(channelUri);
    }
      OnSubscribed(channelUri, false);
}
#endregion
```

17. Créez une méthode **OnSubscribed** comme indiqué ci-dessous. Cette fonction déclenche l'évènement Subscribed:

(Code Snippet – *Using Push Notifications – Registration Service – OnSubscribed function region*)

C#

```
#region Helper private functionality
private static void OnSubscribed(Uri channelUri, bool isActive)
{
    EventHandler<SubscriptionEventArgs> handler = Subscribed;
    if (handler != null)
    {
       handler(null,
                   new SubscriptionEventArgs(channelUri,
                                             isActive));
    }
}
#endregion
```

18. L'évènement utilise SubscriptionEventArgs qui contient les arguments relatifs à l'évènement, comme l'URI et son statut (pour mettre à jour l'interface graphique). Créez la classe (internal) **SubscriptionEventArgs**, dans la classe **RegistrationService**:

(Code Snippet – *Using Push Notifications – Registration Service – SubscriptionEventArgs internal class region*)

C#

```
#region Internal SubscriptionEventArgs class definition
public class SubscriptionEventArgs : EventArgs
{
    public SubscriptionEventArgs(Uri channelUri, bool isActive)
    {
        this.ChannelUri = channelUri;
        this.IsActive = isActive;
    }

    public Uri ChannelUri { get; private set; }
    public bool IsActive { get; private set; }
}
#endregion
```

19. Enfin, créez une méthode publique qui renvoie la liste de tous les abonnés ; celle-ci sera utilisée plus tard par l'application WPF.

(Code Snippet – *Using Push Notifications – Registration Service – GetSubscribers helper function region*)

C#

```
#region Helper public functionality
public static List<Uri> GetSubscribers()
{
    return subscribers;
}
#endregion
```

20. Ouvrez le fichier **App.xaml.cs**.

21. Localisez la ligne avec "*//TODO - remove remark after creating registration service*" et supprimez la remarque de l'initialisation de l'hôte du service WCF :

C#

```
//TODO - remove remark after creating registration service
host = new ServiceHost(typeof(RegistrationService));
host.Open();
```

22. Compilez et exécutez l'application. Lorsque celle-ci a démarrée, vérifiez que le service WCF RESTful fonctionne. Pour cela, ajoutez un point d'arrêt dans les méthodes **Register** et/ou **Unregister** de **RegistrationService.cs**.

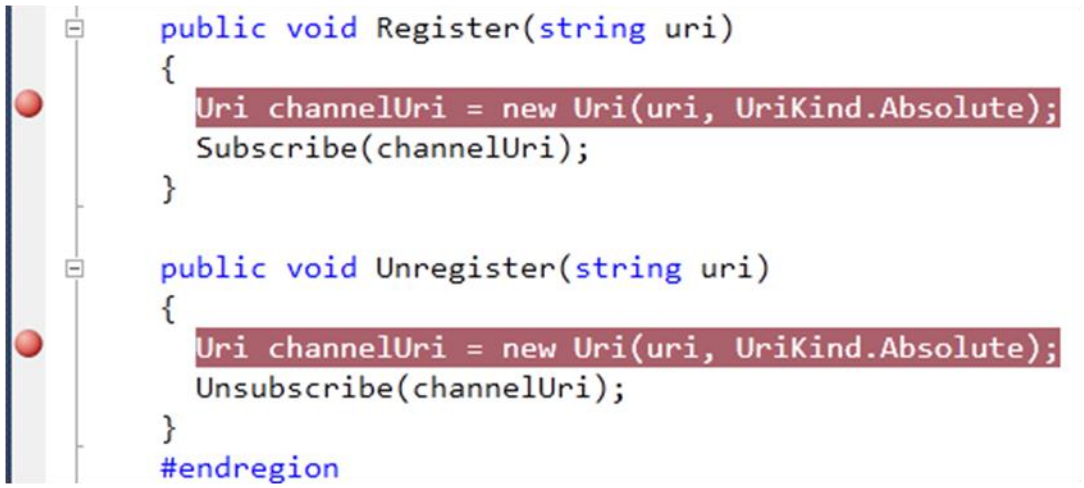


Figure 12

Points d'arrêt pour vérifier le service WCF RESTful

23. Lancez Internet Explorer et naviguez à l'adresse http://localhost:8000/RegirstatorService/Register?uri=http://www.microsoft.com

Nous utilisons une URI qui est codée en dur et mappée dans le fichier de configuration du service WCF REST. Cela permet de simplifier l'atelier.

24. Lorsque le debugger s'arrête sur le point d'arrêt, vérifiez que l'URI est bien http://www.microsoft.com.

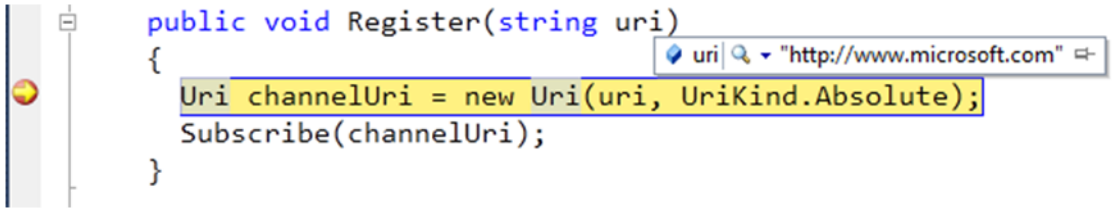


Figure 13

Vérification du service WCF RESTful

25. Arrêtez le debugger, enlevez les points d'arrêt et fermez Internet Explorer.

    Afin de communiquer avec le service MPN, vous allez créer une classe d'utilitaire qui se chargera des charges complexes à votre place. Cette classe contiendra tout la logique de communication nécessaire pour envoyer des messages au serveur MPN.

26. Ajoutez le projet **NotificationSenderUtility** depuis le répertoire **Source\Assets** de cet atelier. Il s'agit d'une bibliothèque de classe vide, qui référence le Framework .NET 4 et non le .NET Framework Client Profile 4.

27. Ouvrez le fichier **NotificationSenderUtility.cs**.

Cette classe gère toute les fonctionnalités nécessaires pour communiquer avec le service MPN. Durant les prochaines étapes, vous allez ajouter des fonctionnalités pour préparer et envoyer des messages au serveur MPN. Cette classe expose aussi une interface publique que l'application WPF utilise pour envoyer des messages aux URIs enregistrées dans la liste. Cette classe utilise le pattern Asynchrone pour communiquer avec le serveur MPN et notifiera la méthode appelante en utilisant les fonctions de callback fournies.

28. Ajoutez l'énumération **NotificationType** à l'espace de nom (juste avant la classe **NotificationSenderUtility**)

(Code Snippet – *Using Push Notifications – NotificationSenderUtility –NotificationType Enum*)

C#

```
#region NotificationType Enum
public enum NotificationType
{
    Token = 1,
    Toast = 2,
    Raw = 3
}
```

```
#endregion
```

29. Puis, suite à la définition de l'énumération (après la classe **NotificationSenderUtility**), ajoutez une classe qui servira à contenir les données reçues du service MPN et à les transférer, via des paramètres, aux callback, sous forme d'un objet de type HttpWebResponse :

(Code Snippet – *Using Push Notifications – NotificationSenderUtility –CallbackArgs class*)

C#

```
#region Event & Event Handler Definition
public class CallbackArgs
{
   public CallbackArgs(NotificationType notificationType,
               HttpWebResponse response)
   {
      this.Timestamp = DateTimeOffset.Now;
      this.MessageId =
response.Headers[NotificationSenderUtility.MESSAGE_ID_HEADER];
      this.ChannelUri = response.ResponseUri.ToString();
      this.NotificationType = notificationType;
      this.StatusCode = response.StatusCode;
      this.NotificationStatus =
response.Headers[NotificationSenderUtility.NOTIFICATION_STATUS_HEADER];
      this.DeviceConnectionStatus =
response.Headers[NotificationSenderUtility.DEVICE_CONNECTION_STATUS_HEAD
ER];
      this.SubscriptionStatus =
response.Headers[NotificationSenderUtility.SUBSCRIPTION_STATUS_HEADER];
   }

   public DateTimeOffset Timestamp { get; private set; }
   public string MessageId { get; private set; }
   public string ChannelUri { get; private set; }
   public NotificationType NotificationType { get; private set; }
   public HttpStatusCode StatusCode { get; private set; }
   public string NotificationStatus { get; private set; }
   public string DeviceConnectionStatus { get; private set; }
   public string SubscriptionStatus { get; private set; }
}
#endregion
```

30. Dans la classe **NotificationSenderUtility**, ajoutez des constantes relatives à la communication avec le service MPN.

(Code Snippet – *Using Push Notifications – NotificationSenderUtility –Push Notification Service communication constants*)

C#

```
#region Local constants
public const string MESSAGE_ID_HEADER = "X-MessageID";
public const string NOTIFICATION_CLASS_HEADER = "X-NotificationClass";
public const string NOTIFICATION_STATUS_HEADER = "X-NotificationStatus";
public const string DEVICE_CONNECTION_STATUS_HEADER = "X-
DeviceConnectionStatus";
public const string SUBSCRIPTION_STATUS_HEADER = "X-SubscriptionStatus";
public const string WINDOWSPHONE_TARGET_HEADER = "X-WindowsPhone-
Target";
public const int MAX_PAYLOAD_LENGTH = 1024;
#endregion
```

31. Créez un délégué qui déclare la signature pour les callback et ajouter une méthode permettant de l'exécuter:

(Code Snippet – *Using Push Notifications – NotificationSenderUtility –Notification Callback Delegate*)

```
C#
#region Notification Callback Delegate Definition
public delegate void SendNotificationToMPNSCompleted(CallbackArgs
response);
#endregion
```

(Code Snippet – *Using Push Notifications – NotificationSenderUtility –Helper function to executed delegated callback function*)

```
C#
#region Callback call
protected void OnNotified(NotificationType notificationType,
             HttpWebResponse response,
      SendNotificationToMPNSCompleted callback)
{
    CallbackArgs args = new CallbackArgs(notificationType, response);
    if (null != callback)
        callback(args);
}
#endregion
```

Par la suite, vous allez créer la fonctionnalité qui est nécessaire pour envoyer des notifications HTTP brutes au serveur MPN. Vous allez donc créer une méthode générique qui prépare et envoie les messages, en fonction du type de notification.

32. Créez tout d'abord une méthode publique nommée **SendRawNotification** pour envoyer des messages de notifications HTTP brutes à tous les clients abonnés. Ajoutez le code suivant dans la classe **NotificationSenderUtility** :

(Code Snippet – *Using Push Notifications – NotificationSenderUtility – SendRawNotifiaction function*)

C#

```
#region SendXXXNotification functionality
public void SendRawNotification(List<Uri> Uris, byte[] Payload,
SendNotificationToMPNSCompleted callback)
{
    foreach (var uri in Uris)
        SendNotificationByType(uri, Payload, NotificationType.Raw,
callback);

}
#endregion
```

33. Puis, ajoutez une fonction **SendNotificationByType**. Il s'agit de la fonction générique qui appellera la fonction d'envoi spécifique:

(Code Snippet – *Using Push Notifications – NotificationSenderUtility – SendNotificationByType function*)

C#

```
#region SendNotificationByType Logic
private void SendNotificationByType(Uri channelUri,
    byte[] payload, NotificationType notificationType,
SendNotificationToMPNSCompleted callback)
{
    try
    {
        SendMessage(channelUri, payload, notificationType, callback);
    }
    catch (Exception ex)
    {
        throw;
    }
}
#endregion
```

Le service de notification de Push de Microsoft est un service dit "dans les nuages", qui est utilisé pour pousser des messages à tous les clients Windows Phone 7 enregistrés. Plus tard dans l'atelier, vous allez créer une application Windows Phone 7 qui ouvre un canal d'envoi des messages vers ce service, qui répond via une URI – une URL unique qui représente l'interface REST à travers laquelle votre service de backoffice peut envoyer des messages. Pour pouvoir pousser des notifications aux périphériques, le service MPN attend un appel Web (une requête Web) sur cette URI avec l'ensemble des données à envoyer. Après réception de la requête, le service MPN localisera le périphérique et enverra les données. Dans les étapes précédentes, vous avez créé la fonctionnalité permettant de faire un appel au service.

La fonction **SendMessage** effectue toute la communication, vérifie la taille des données et déclenche une exception si celle-ci est trop importante. Si tout est OK, elle utilise la

classe **HttpWebRequest** (de l'espace de nom System.Net. Plus d'infos sur : http://msdn.microsoft.com/en-us/library/system.net.httpwebrequest.aspx) et écrit les informations dans le flux du canal. Après avoir envoyé les informations, la fonction attend une réponse du service MPN et ensuite, notifie l'appelant en réappelant le délégué.

34. Ajoutez la fonction suivante, qui enverra la requête au service MPN :

(Code Snippet – *Using Push Notifications – NotificationSenderUtility –SendMessage function*)

C#

```
#region Send Message to Microsoft Push Service
private void SendMessage(Uri channelUri, byte[] payload,
NotificationType notificationType, SendNotificationToMPNSCompleted
callback)
{
   //Check the length of the payload and reject it if too long
   if (payload.Length > MAX_PAYLOAD_LENGTH)
      throw new ArgumentOutOfRangeException(
"Payload is too long. Maximum payload size shouldn't exceed " +
MAX_PAYLOAD_LENGTH.ToString() + " bytes");

   try
   {
      //TODO - send message to MPNS
   }
   catch (WebException ex)
   {
      if (ex.Status == WebExceptionStatus.ProtocolError)
      {
         //Notify client on exception
         OnNotified(notificationType,
             (HttpWebResponse)ex.Response, callback);
      }
      throw;
   }
}
#endregion
```

35. Dans le bloc try, localisez "//TODO - send message to MPNS" et remplacez-le par l'initialisation de **HttpWebRequest** :

(Code Snippet – *Using Push Notifications – NotificationSenderUtility –HttpWebRequest initializations*)

C#

```
//Create and initialize the request object
```

```
HttpWebRequest request =
      (HttpWebRequest)WebRequest.Create(channelUri);
request.Method = WebRequestMethods.Http.Post;
request.ContentType = "text/xml; charset=utf-8";
request.ContentLength = payload.Length;
request.Headers[MESSAGE_ID_HEADER] =   Guid.NewGuid().ToString();
request.Headers[NOTIFICATION_CLASS_HEADER] =
      ((int)notificationType).ToString();

if (notificationType == NotificationType.Toast)
   request.Headers[WINDOWSPHONE_TARGET_HEADER] = "toast";
else if (notificationType == NotificationType.Token)
   request.Headers[WINDOWSPHONE_TARGET_HEADER] = "token";
```

Lorsque la requête est prête, vous devez l'ouvrir de façon asynchrone. Lorsque le flux est ouvert, vous aurez accès à l'objet Stream (de l'espace de nom System.IO; plus d'infos: http://msdn.microsoft.com/en-us/library/system.io.stream.aspx). Cet objet de type Stream représente le flux de la requête sur lequel les données seront écrites de façon asynchrone. Après qu'elle ait fini sa procédure d'écriture, la fonction attendra la réponse du service MPN. Lorsque la réponse arrivera, la fonction notifiera l'appelant en utilisant les délégués.

36. Ajoutez l'extrait de code suivant au corps de la méthode :

(Code Snippet – *Using Push Notifications – NotificationSenderUtility –MPNS call*)

C#

```
request.BeginGetRequestStream((ar) =>
{
   //Once async call returns get the Stream object
   Stream requestStream = request.EndGetRequestStream(ar);

   //and start to write the payload to the stream asynchronously
   requestStream.BeginWrite(payload, 0, payload.Length,    (iar) =>
     {
        //When the writing is done, close the stream
        requestStream.EndWrite(iar);
        requestStream.Close();

        //and switch to receiving the response from MPNS
        request.BeginGetResponse((iarr) =>
        {
           using (WebResponse response =
             request.EndGetResponse(iarr))
           {
              //Notify the caller with the MPNS results
              OnNotified(notificationType,
             (HttpWebResponse)response, callback);
           }
```

```
        },
        null);
    },
    null);
},
null);
```

37. Sauvez et fermez la classe **NotificationSenderUtility**.

38. Depuis l'application Weather, ajoutez une référence à la bibliothèque de classe **NotificationSenderUtility** en faisant un clic droit sur **References** (du projet **Weather**) et en sélectionnant **Add Reference**.

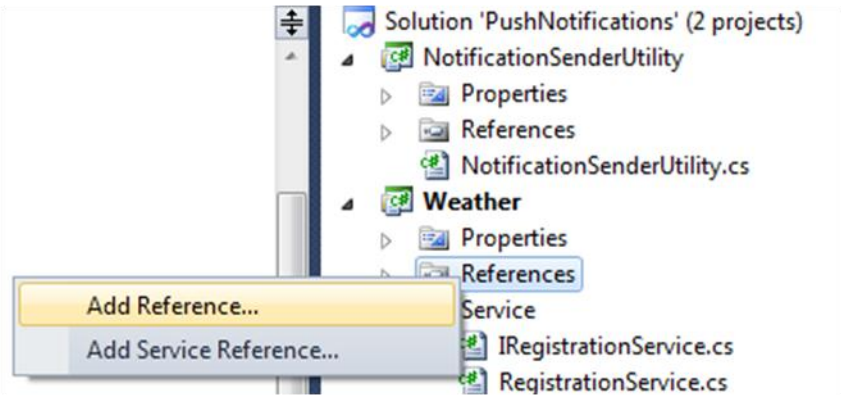


Figure 14

Ajout d'une référence

39. Dans la boite de dialogue, cliquez sur l'onglet **Projects**, sélectionnez **NotificationSenderUtility** et cliquez sur OK.

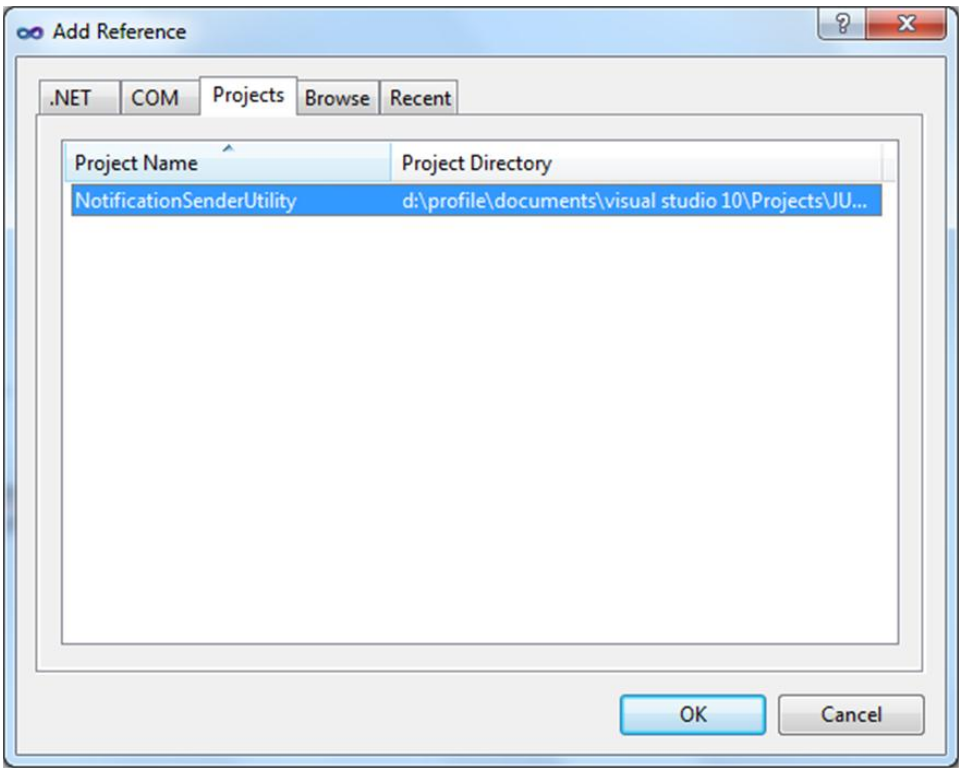


Figure 15

Ajout d'une référence

40. Ouvrez **MainWindows.xaml.cs**.

41. Ajoutez les "using" suivants :

(Code Snippet – *Using Push Notifications – MainWindow.xaml.cs –using statements*)

C#

```
using System.Collections.ObjectModel;
using System.Threading;
using System.IO;
using System.Xml;
using WindowsPhone.PushNotificationManager;
using WeatherService.Service;
```

42. Localisez la region **Private variables** et remplacez le contenu par ce code:

(Code Snippet – *Using Push Notifications – MainWindow.xaml.cs –Private variables*)

C#

```
private ObservableCollection<CallbackArgs> trace = new
      ObservableCollection<CallbackArgs>();
private NotificationSenderUtility notifier = new
      NotificationSenderUtility();
private string[] lastSent = null;
```

Notre application cliente Windows Phone 7 a besoin de logique pour souscrire au service précédemment créé. Notre application WPF simule l'envoi d'informations de temps à tous les clients connectés ; elle doit donc connaitre les applications clientes de type Windows Phone 7 et être capable de recevoir des évènements d'enregistrement.

43. Localisez le constructeur de la classe et ajoutez ce code, après le commentaire "TODO":

(Code Snippet – *Using Push Notifications – MainWindow.xaml.cs –ctor additional initializations*)

C#

```
Log.ItemsSource = trace;
RegistrationService.Subscribed += new
EventHandler<RegistrationService.SubscriptionEventArgs>(RegistrationServ
ice_Subscribed);
```

44. Trouvez la région **Event Handlers** et ajoutez la fonction suivante :

(Code Snippet – *Using Push Notifications – MainWindow.xaml.cs – RegistrationService_Subscribed function*)

C#

```
void RegistrationService_Subscribed(object sender,
RegistrationService.SubscriptionEventArgs e)
{
   //Check previous notifications, and resend last one to connected client

   Dispatcher.BeginInvoke((Action)(() =>
             { UpdateStatus(); })
      );
}
```

45. Après cette méthode, ajoutez celle-ci qui sert de callback pour les appels à la classe **NotificationSenderUtility**:

(Code Snippet – *Using Push Notifications – MainWindow.xaml.cs –OnMessageSent function*)

C#

```
private void OnMessageSent(CallbackArgs response)
{
   Dispatcher.BeginInvoke((Action)(() => {
      trace.Add(response); }));
}
```

46. Trouvez la région **Private functionality** et ajoutez ce code (cette méthode met à jour l'interface graphique de l'application WPF) :

(Code Snippet – *Using Push Notifications – MainWindow.xaml.cs –UpdateStatus function*)

C#

```
private void UpdateStatus()
{
   int activeSubscribers =
       RegistrationService.GetSubscribers().Count;
   bool isReady = (activeSubscribers > 0);
   txtActiveConnections.Text = activeSubscribers.ToString();
   txtStatus.Text = isReady ? "Ready" : "Waiting for connection...";
   btnSendHttp.IsEnabled = isReady;
   btnSendTile.IsEnabled = isReady;
   btnSendToast.IsEnabled = isReady;
}
```

47. Localisez le gestionnaire d'évènement **btnSendHttp_Click** et ajoutez ce code. Cette méthode servira à lister tous les clients connectés, à préparer les données depuis la saisie dans l'interface graphique et à les envoyer grâce à un appel asynchrone de la classe NotificationSenderUtility, effectué au moyen du ThreadPool (de l'espace de nom System.Threading. Plus d'infos : http://msdn.microsoft.com/en-us/library/system.threading.threadpool.aspx):

(Code Snippet – *Using Push Notifications – MainWindow.xaml.cs –btnSendHttp_Click function body*)

C#

```
//Get the list of subscribed WP7 clients
List<Uri> subscribers =   RegistrationService.GetSubscribers();
//Prepare payload
byte[] payload = prepareRAWPayload(
      cmbLocation.SelectedValue as string,
      sld.Value.ToString("F1"),
      cmbWeather.SelectedValue as string);
//Invoke sending logic asynchronously
ThreadPool.QueueUserWorkItem((unused) =>
      notifier.SendRawNotification(subscribers,
                                   payload,
                                   OnMessageSent)
      );

//Save last RAW notification for future usage
lastSent = new string[3];
lastSent[0] = cmbLocation.SelectedValue as string;
lastSent[1] = sld.Value.ToString("F1");
lastSent[2] = cmbWeather.SelectedValue as string;
```

Note: Dans notre cas spécifique, la classe NotificationSenderUtility implémente un pattern de communication asynchrone compris par HttpWebRequest. Ainsi, il n'est absolument pas nécessaire d'invoquer la fonction via le ThreadPool (asynchrone), mais d'une manière générale, c'est une bonne façon de faire que d'invoquer la fonctionnalité de communication de telle sorte que les mises à jour de l'interface graphique soient découplées d'une opération potentiellement longue. La communication est un processus long et, si elle ne supporte pas l'asynchronisme, un simple appel pourrait geler l'interface graphique.

48. Trouvez la région **Private functionality** et ajoutez-y la méthode **prepareRAWPayload**. Cette fonction créera le document XML en mémoire et le renverra sous forme de tableau d'octet (prêt à être envoyé aux applications clientes Windows Phone 7) :

(Code Snippet – *Using Push Notifications – MainWindow.xaml.cs –PrepareRAWPayload function*)

C#

```
private static byte[] prepareRAWPayload(string location,
       string temperature, string weatherType)
{
   MemoryStream stream = new MemoryStream();

   XmlWriterSettings settings = new XmlWriterSettings()
       { Indent = true, Encoding = Encoding.UTF8 };
   XmlWriter writer = XmlTextWriter.Create(stream,
                           settings);

   writer.WriteStartDocument();
   writer.WriteStartElement("WeatherUpdate");

   writer.WriteStartElement("Location");
   writer.WriteValue(location);
   writer.WriteEndElement();

   writer.WriteStartElement("Temperature");
   writer.WriteValue(temperature);
   writer.WriteEndElement();

   writer.WriteStartElement("WeatherType");
   writer.WriteValue(weatherType);
   writer.WriteEndElement();

   writer.WriteStartElement("LastUpdated");
   writer.WriteValue(DateTime.Now.ToString());
   writer.WriteEndElement();
```

```
    writer.WriteEndElement();
    writer.WriteEndDocument();
    writer.Close();

    byte[] payload = stream.ToArray();
    return payload;
}
```

49. Localisez la méthode **RegistrationService_Subscribed**, et ajoutez ce code, qui renvoie les messages bruts aux clients connectés avant d'appeler la méthode **UpdateStatus**:

(Code Snippet – *Using Push Notifications – MainWindow.xaml.cs – Resend RAW message*)

C#

```
if (null != lastSent)
{
    string location = lastSent[0];
    string temperature = lastSent[1];
    string weatherType = lastSent[2];
    List<Uri> subscribers = new List<Uri>();
    subscribers.Add(e.ChannelUri);
    byte[] payload = prepareRAWPayload(location, temperature,
        weatherType);

    ThreadPool.QueueUserWorkItem((unused) =>
        notifier.SendRawNotification(subscribers, payload,
        OnMessageSent));
}
```

50. Compilez l'application et corrigez les erreurs de compilation (s'il y en a)

Etape 2 – Création de l'application Windows Phone 7

1. Ajoutez un nouveau projet de type Windows Phone Application à la solution et nommez le : **PushNotifications**.

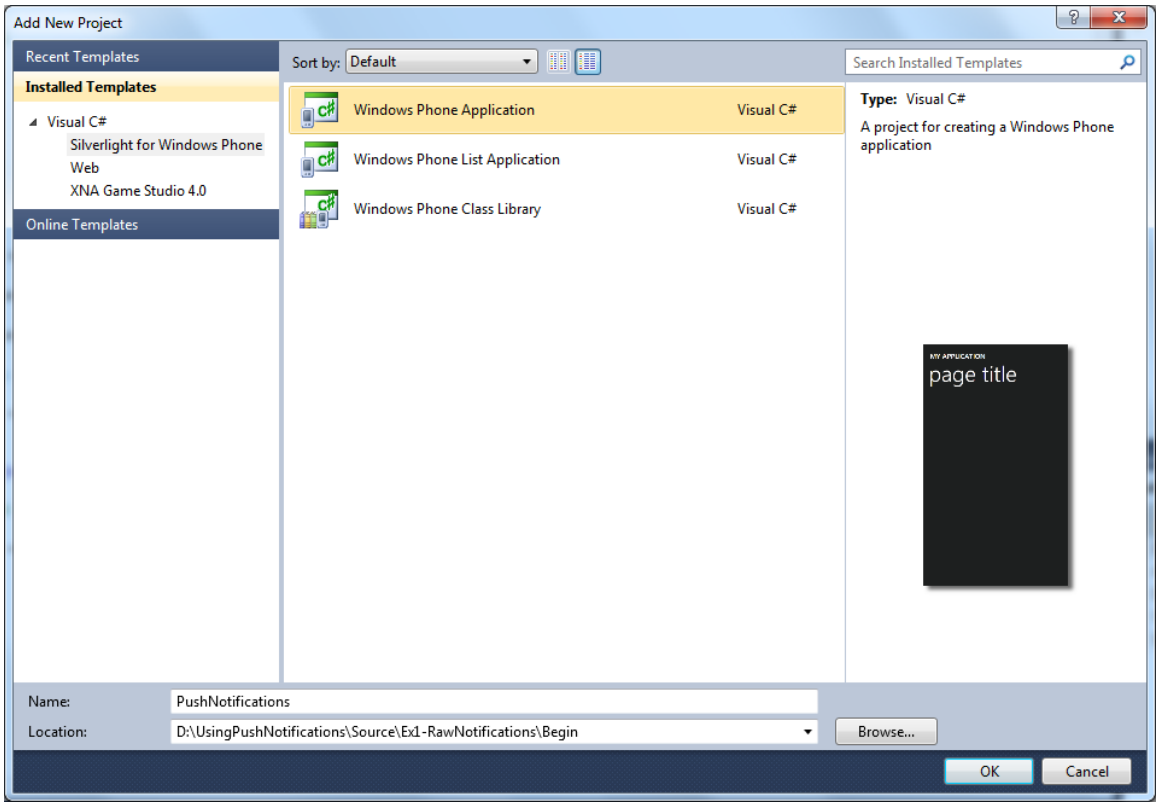


Figure 16

Ajout d'un projet de type Windows Phone Application à la solution

2. Ajoutez une référence à l'assembly **Microsoft.Phone.Notification** (de l'onglet .NET)

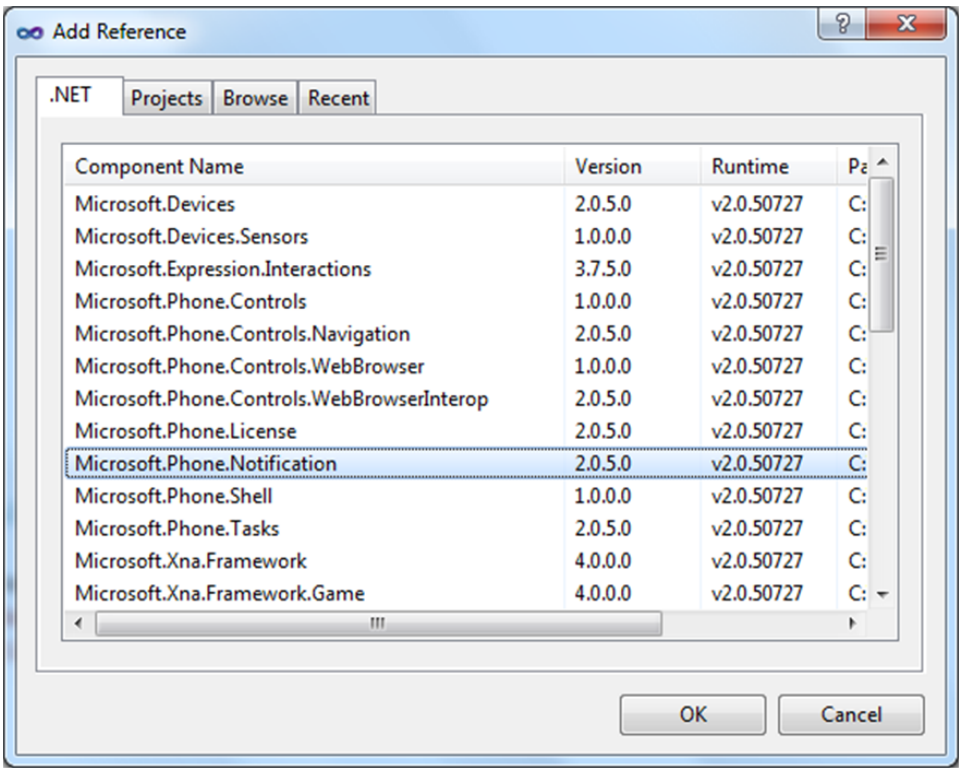


Figure 17

Ajout d'une référence à Microsoft.Phone.Notification

3. Répétez l'étape précédente et ajoutez une référence à l'assembly **System.Xml.Linq**:

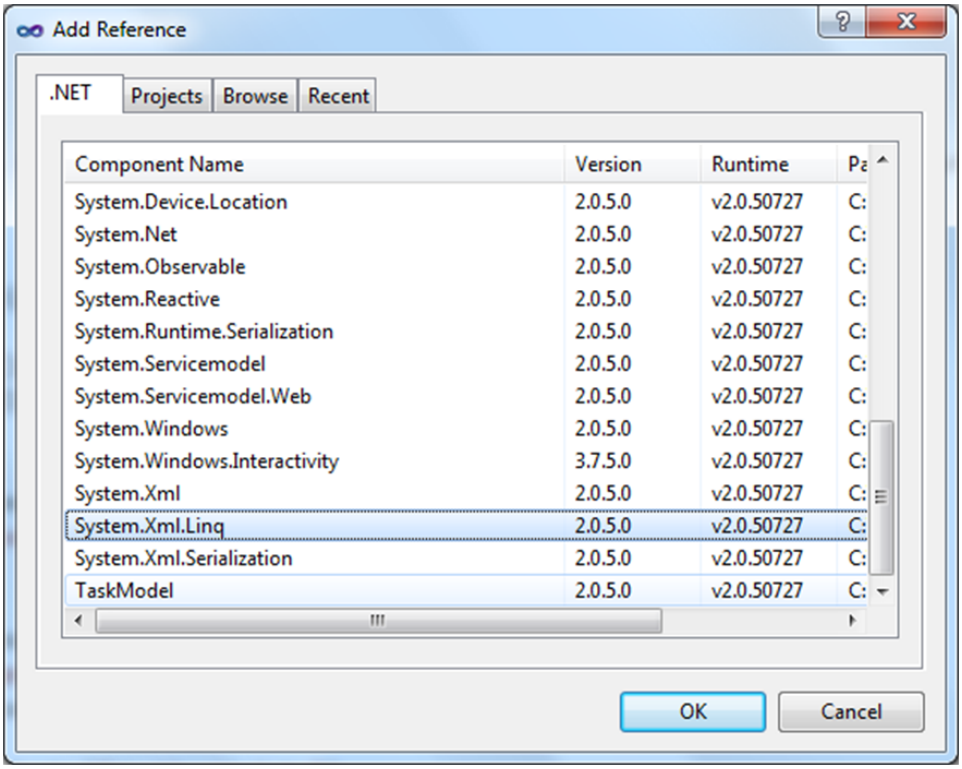


Figure 18

Ajout d'une référence à System.Xml.Linq

4. Ouvrez **MainPage.xaml** (s'il ne s'est pas ouvert automatiquement).

5. Localisez la grille **LayoutRoot** et remplacez son contenu par cet extrait de code:

XAML

```
<Grid.RowDefinitions>
   <RowDefinition Height="Auto"/>
   <RowDefinition Height="*"/>
   <RowDefinition Height="150"/>
   <RowDefinition Height="30"/>
</Grid.RowDefinitions>

<!--TitleGrid is the name of the application and page title-->
   <Grid x:Name="TitleGrid" Grid.Row="0">
      <TextBlock Text="WEATHER SERVICE"
       x:Name="textBlockPageTitle"
       Style="{StaticResource PhoneTextPageTitle1Style}"/>
     <TextBlock x:Name="textBlockListTitle"   Style="{StaticResource
PhoneTextPageTitle2Style}"/>
   </Grid>

   <!--ContentGrid is empty. Place new content here-->
   <StackPanel x:Name="ContentStackPanel" Grid.Row="1">
      <TextBlock Text="Current Conditions"
       HorizontalAlignment="Center"
       Style="{StaticResource PhoneTextGroupHeaderStyle}" />
     <Grid HorizontalAlignment="Center" Width="150"
       Height="150">
         <Border BorderBrush="LightGray" BorderThickness="1">
            <Image x:Name="imgWeatherConditions" Width="150"
              Height="150" Stretch="None"
       VerticalAlignment="Top"
       HorizontalAlignment="Center"/>
         </Border>
         <TextBlock x:Name="txtTemperature"
       HorizontalAlignment="Center"
VerticalAlignment="Bottom"
        Style="{StaticResource PhoneTextExtraLargeStyle}" />
      </Grid>
   </StackPanel>

   <StackPanel Grid.Row="3" Orientation="Vertical"
      x:Name="StatusStackPanel" >
      <StackPanel Orientation="Horizontal" Opacity=".5">
         <TextBlock Text="Status: "
             Style="{StaticResource PhoneTextNormalStyle}"/>
```

```
        <TextBlock x:Name="txtStatus"
      Style="{StaticResource PhoneTextNormalStyle}"
      Text="Not Connected"/>
     </StackPanel>
   </StackPanel>
```

6. Ouvrez **MainPage.xaml.cs**.

7. Ajoutez les using suivants:

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs –using statements*)

C#

```
using Microsoft.Phone.Notification;
using System.Diagnostics;
using System.Windows.Threading;
using System.Windows.Media.Imaging;
using System.IO;
using System.Xml.Linq;
using System.IO.IsolatedStorage;
```

8. Ajoutez ce bout de code:

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs –private variables*)

C#

```
private HttpNotificationChannel httpChannel;
const string channelName = "WeatherUpdatesChannel";
const string fileName = "PushNotificationsSettings.dat";
const int pushConnectTimeout = 120;
```

9. Ajoutez ce bout de code, contenant des méthodes d'aide. Ces méthodes mettront à jour l'interface graphique de l'application Windows Phone 7 et afficheront des informations de trace utiles:

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs –Updating and Tracing functions*)

C#

```
#region Tracing and Status Updates
private void UpdateStatus(string message)
{
   txtStatus.Text = message;
}

private void Trace(string message)
{
#if DEBUG
   Debug.WriteLine(message);
#endif
}
```

```
#endregion
```

10. Définissez **PushNotifications** comme projet de démarrage. Pour cela, faîtes un clic droit sur le nom du projet et sélectionnez **Set as StartUp Project**.

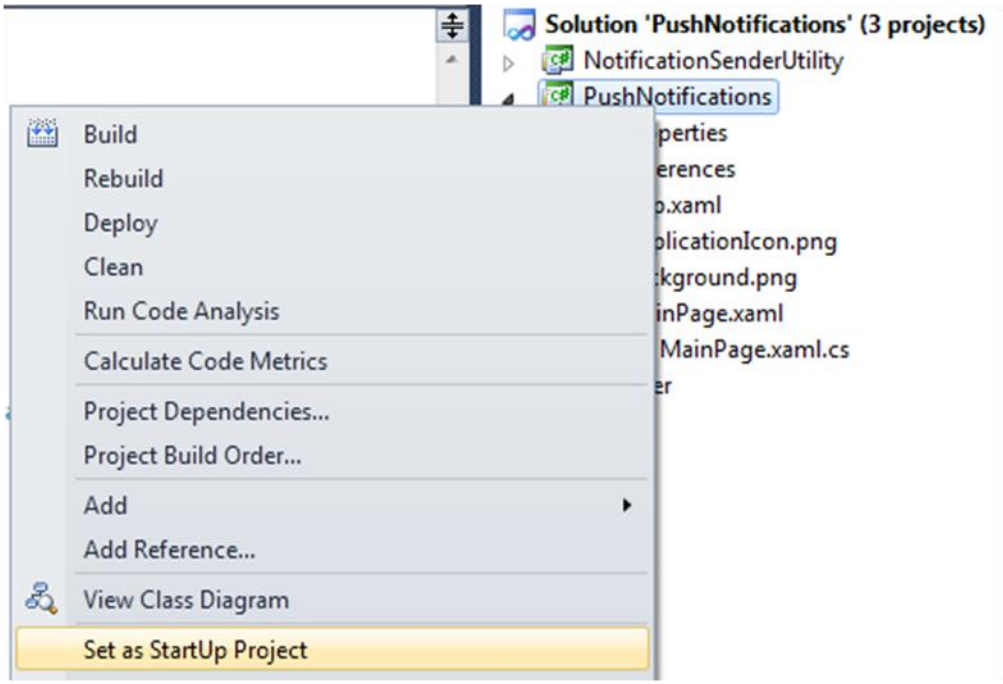


Figure 19

Définition de PushNotifications comme projet de démarrage

11. Compilez et exécutez l'application

12. Pour le moment, l'application doit ressembler à cela :

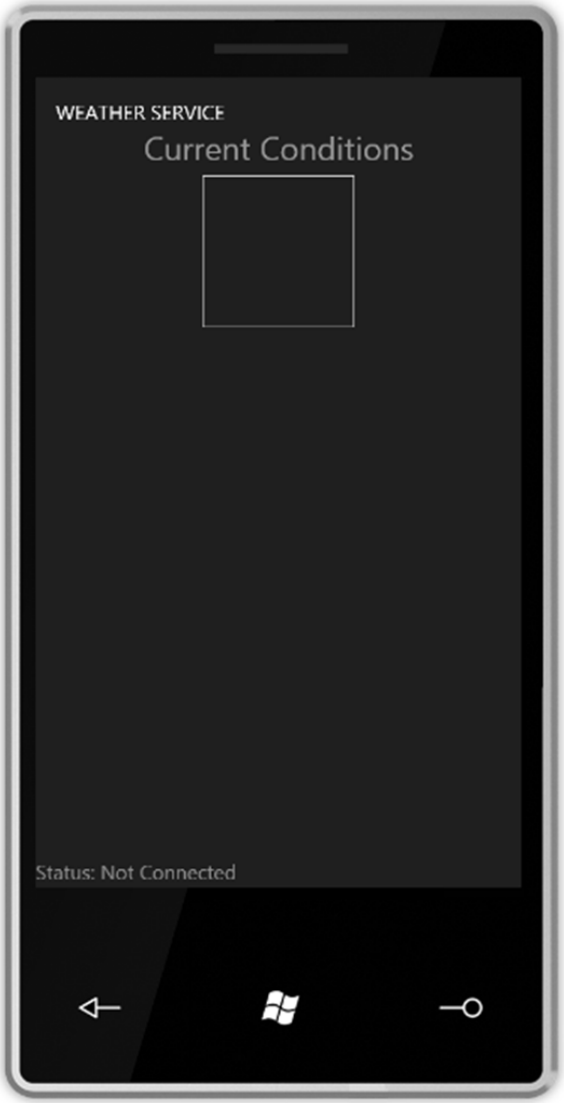


Figure 20

Exécution de l'application

13. Arrêtez le débogage et retournez dans le code

Etape 3 – Création d'un canal de notification et souscription aux évènements du canal

1. Ouvrez **MainPage.xaml.**cs du projet **PushNotifications** (si le fichier était fermé).

   Durant les prochaines étapes, vous allez créer un nouveau canal et souscrire à ces évènements. **HttpNotificationChannel** est la classe qui créée un canal de notification entre le service MPN et l'application cliente, et qui créé une nouvelle souscription à chaque notification. La logique du canal est la suivante:

- Si le canal est déjà connu alors l'application cliente tentera de le rouvrir. Essayer de recréer un canal déjà existant déclenche une exception.
- Si le canal n'est pas déjà ouvert, alors on s'abonne à ces évènements et on essaye de l'ouvrir. Si le canal s'ouvre, cela déclenche l'évènement **ChannelUriUpdated**. Cet évènement signale au client que le canal a été correctement créé.
- Un canal déjà existant peut-être trouvé par son nom. Si la recherche est un succès, il sera réactivé et pourra être utilisé par l'application.

Note : Il y a un bug dans la version actuelle de l'émulateur qui cause une exception lorsque vous essayez d'ouvrir un nouveau canal immédiatement après le démarrage de l'émulateur. Ce bug empêche l'émulateur d'ouvrir un canal valide pendant les 120-180 secondes qui suivent le démarrage de l'émulateur, et survient à chaque fois que celui-ci est démarré. Comme solution de contournement dans l'atelier, nous allons attendre l'exception spécifique et retarder l'exécution de l'application de 180 secondes. **Merci de ne pas fermer l'émulateur pendant le démarrage de l'application.**

2. Créez une nouvelle région, comme le montre ce bout de code:

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs –DoConnect function*)

C#

```
#region Misc logic
private void DoConnect()
{
    //TODO - place connection logic here
}
#endregion
```

3. Créez un bloc Try/Catch dans le corps de la fonction :

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs –DoConnect try-catch block in function body*)

C#

```
try
{

}
catch (NotificationChannelExistsException)
{
    //Catch existing channel exception
}
catch (NotificationChannelOpenException)
{
    //Catch open channel exception - emulator bug workaround
}
```

```
catch (Exception ex)
{
   UpdateStatus("Channel error: " + ex.Message);
}
```

4. Ensuite, initialisez une variable représentant votre canal, abonnez-vous à ces évènements et essayez de l'ouvrir :

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs –DoConnect try block body*)

C#

```
//Catch open channel exception - emulator bug workaround
Trace("Creating new channel");
//Create the channel
httpChannel = new HttpNotificationChannel(channelName,
       "HOLWeatherService");

Trace("Subscribing to channel events");
//Register to UriUpdated event - occurs when channel successfully opens
httpChannel.ChannelUriUpdated += new
EventHandler<NotificationChannelUriEventArgs>(httpChannel_ChannelUriUpda
ted);

SubscribeToChannelEvents();

Trace("Opening channel");
httpChannel.Open();
UpdateStatus("Channel open requested");
```

5. Si l'application reçoit un **NotificationChannelExistsException**, alors elle doit essayer de trouver, par son nom, un canal existant et, si le canal est trouvé, l'utiliser. Autrement, elle doit fermer le canal pour libérer l'URI :

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs –DoConnect first catch block body*)

C#

```
Trace("Channel exists! Trying to retrieve by ChannelName");
//Try to find an existing channel
httpChannel = HttpNotificationChannel.Find(channelName);

 if (null != httpChannel.ChannelUri)
 {
    Trace("Got existing channel");
    SubscribeToChannelEvents();

    Trace("Subscribing to channel events");
    SubscribeToService();
```

```
    UpdateStatus("Channel recovered");
}
else
{
   Trace("Channel not found...");
   httpChannel.Close();
   UpdateStatus("The channel was closed prematurely");
}
```

6. Si l'exécution échoue après le démarrage de l'émulateur, ou s'il y a une erreur générale lors de l'ouverture du canal, l'application doit se mettre en pause durant une durée prédéterminée avant de réessayer :

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs –DoConnect second catch block body*)

C#

```
Trace("EMU workaround - wait for " + pushConnectTimeout.ToString() + "
seconds to allow emulator connect to the Cloud Push Service");
//Wait for emulator to connect to the push service and try-again -
EMULATOR ONLY!!!
int elapsedSeconds = pushConnectTimeout;
DispatcherTimer timer = new DispatcherTimer();
timer.Interval = new TimeSpan(0, 0, 0, 1, 0);
timer.Tick += (sender, args) =>
{
   Dispatcher.BeginInvoke(() => UpdateStatus("Waiting for push service
for " + elapsedSeconds.ToString() + " seconds..."));
   if (--elapsedSeconds == 0)
   {
      timer.Stop();
      Dispatcher.BeginInvoke(() => UpdateStatus("Please restart the
application..."));
      Trace("EMU workaround - finish waiting. Restart the
application!");
   }
};
timer.Start();
```

7. Créez une méthode d'aide pour souscrire aux évènements du canal :

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs – SubscribeToChannelEvents function*)

C#

```
#region Subscriptions
private void SubscribeToChannelEvents()
{
   //Subscribe to the channel events
```

```
   httpChannel.HttpNotificationReceived += new
EventHandler<HttpNotificationEventArgs>(httpChannel_HttpNotificationRece
ived);
   httpChannel.ExceptionOccurred += new
EventHandler<NotificationChannelExceptionEventArgs>(httpChannel_Exceptio
nOccurred);
}
#endregion
```

8. Ajoutez une fonction qui va enregistrer le client Windows Phone 7 auprès du service MPN en enregistrant l'URI du canal reçu auprès du service créé précédemment. Cette fonction utilise le WebClient et code en dur (dans notre cas) l'emplacement du service WCF RESTful pour enregistrer l'URI du client. Ajoutez le code suivant dans la région **Subscriptions** :

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs – subscriptions region*)

C#

```
private void SubscribeToService()
{
  //Hardcode for solution - need to be updated in case the REST WCF service
address change
   string baseUri =
"http://localhost:8000/RegirstatorService/Register?uri={0}";
   string theUri = String.Format(baseUri,
       httpChannel.ChannelUri.ToString());
   WebClient client = new WebClient();
   client.DownloadStringCompleted += (s, e) =>
   {
      if (null == e.Error)
         Dispatcher.BeginInvoke(() =>
      UpdateStatus("Registration succeeded"));
      else
         Dispatcher.BeginInvoke(() =>
      UpdateStatus("Registration failed: " +
      e.Error.Message));
   };
   client.DownloadStringAsync(new Uri(theUri));
}
```

9. Créez un gestionnaire d'évènement qui gérera l'évènement **ChannelUriUpdate**. Cet évènement est déclenché lorsque le système WP renvoi l'URI nécessaire depuis le MPN :

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs – httpChannel_ChannelUriUpdated function*)

C#

```
#region Channel event handlers
```

```
void httpChannel_ChannelUriUpdated(object sender,
      NotificationChannelUriEventArgs e)
{
   Trace("Channel opened. Got Uri:\n" +
      httpChannel.ChannelUri.ToString());

    Trace("Subscribing to channel events");
    SubscribeToService();

    Dispatcher.BeginInvoke(() => UpdateStatus("Channel created
successfully"));
}
#endregion
```

10. Ajouter un gestionnaire d'évènement **ExceptionOccured**. Si cette exception est déclenchée, cela signifie qu'une erreur s'est produite : vous devrez arrêter et redémarrer l'émulateur :

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs – httpChannel_ExceptionOccured function*)

C#

```
void httpChannel_ExceptionOccurred(object sender,
      NotificationChannelExceptionEventArgs e)
{
   UpdateStatus("Error occurred: " + e.Exception.Message);
}
```

11. Enfin, ajoutez le gestionnaire d'évènement **HttpNotificationReceived**:

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs – httpChannel_HttpNotificationReceived function*)

C#

```
void httpChannel_HttpNotificationReceived(object sender,
      HttpNotificationEventArgs e)
{

   Trace("==============================================");
   Trace("RAW notification arrived:");

   //TODO - add parsing and UI updating logic here

   Trace("==============================================");
}
```

Cette méthode doit parser les données XML reçues depuis le service MPN et mettre à jour l'interface graphique de l'application cliente.

12. Ajoutez un appel à la méthode **DoConnect** dans le constructeur de la classe. Le code définitif du constructeur doit ressembler à ceci :

C#

```
public MainPage()
{
   InitializeComponent();

   SupportedOrientations = SupportedPageOrientation.Portrait      |
SupportedPageOrientation.Landscape;

   DoConnect();
}
```

13. Compilez le projet et corrigez les erreurs de compilations (s'il y en a)

14. Définissez plusieurs projets de démarrage dans cette solution pour que l'application WPF et le client Windows Phone 7 démarrent ensemble. Pour cela, faîtes un clic droit sur la solution et sélectionnez Properties dans le menu contextuel :

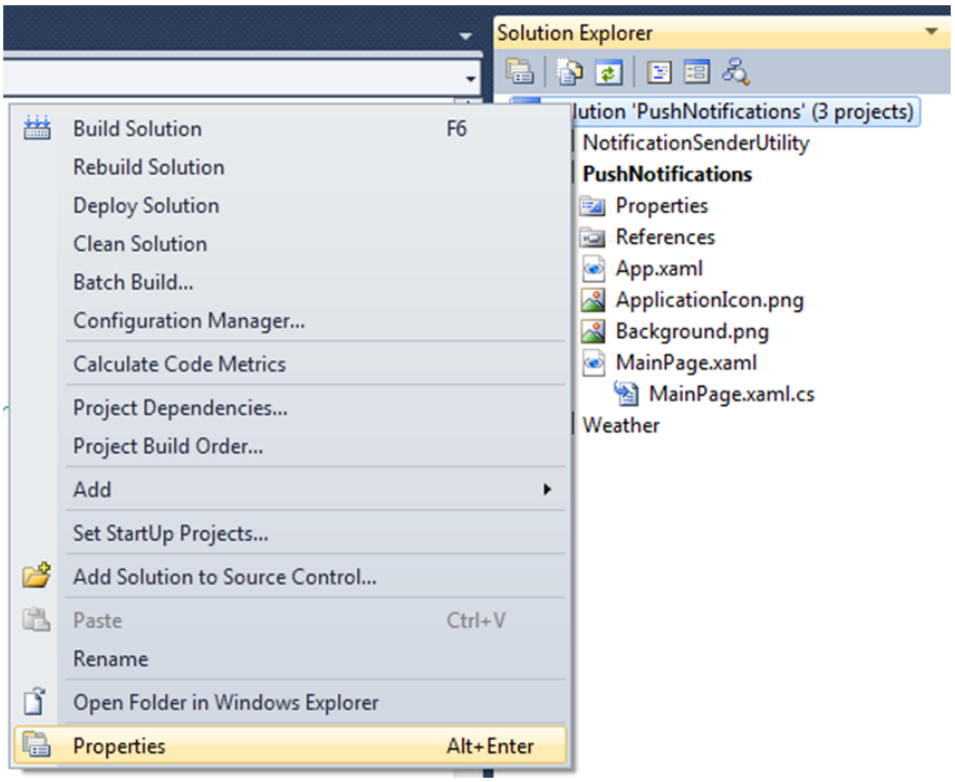


Figure 21

Ouverture des propriétés de la solution

15. Sélectionnez:

- **Startup Projects** de **Common Properties**

- **Multiple startup projects**, puis
- **Start** de la ComboBox **Action** pour les projets **PushNotifications** et **Weather**

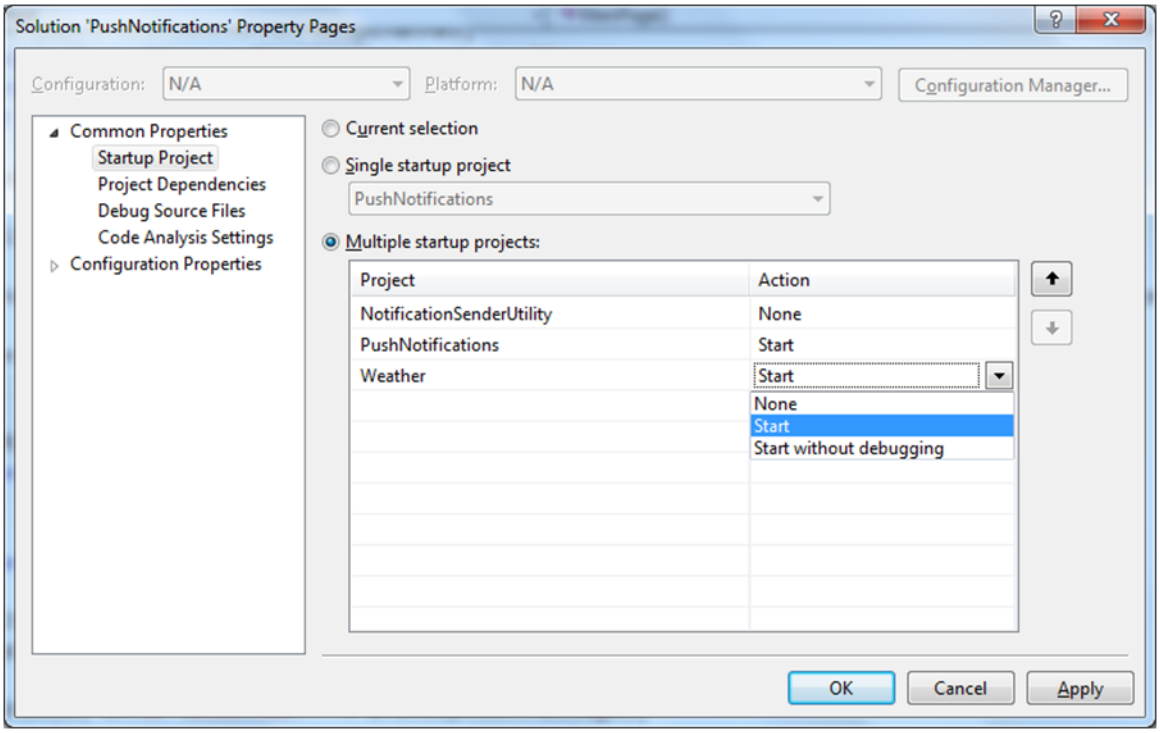


Figure 22

Sélection de plusieurs projets de démarrage

16. Compilez et exécutez l'application

17. Une fois tous les projets démarrés, votre écran doit ressembler à ceci :

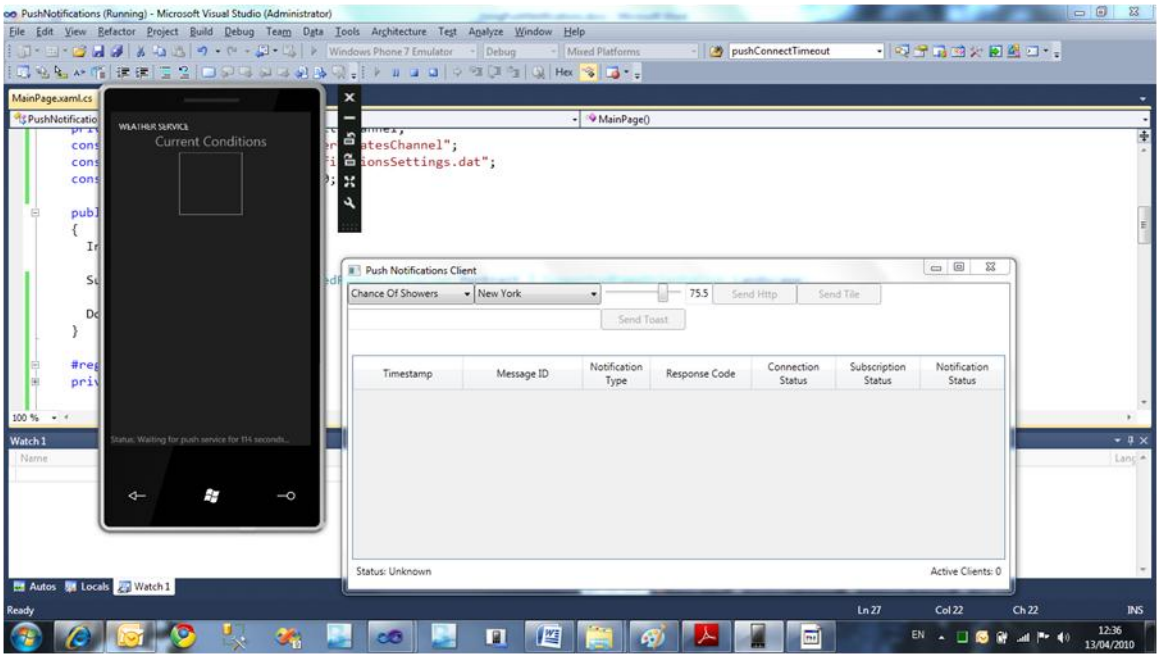


Figure 23

Exécution de plusieurs projets

18. L'instance de l'émulateur du téléphone, au premier démarrage, affiche un compteur en bas de l'écran. Attendez qu'il finisse de compter - (***ne fermez pas la fenêtre de l'émulateur !***).

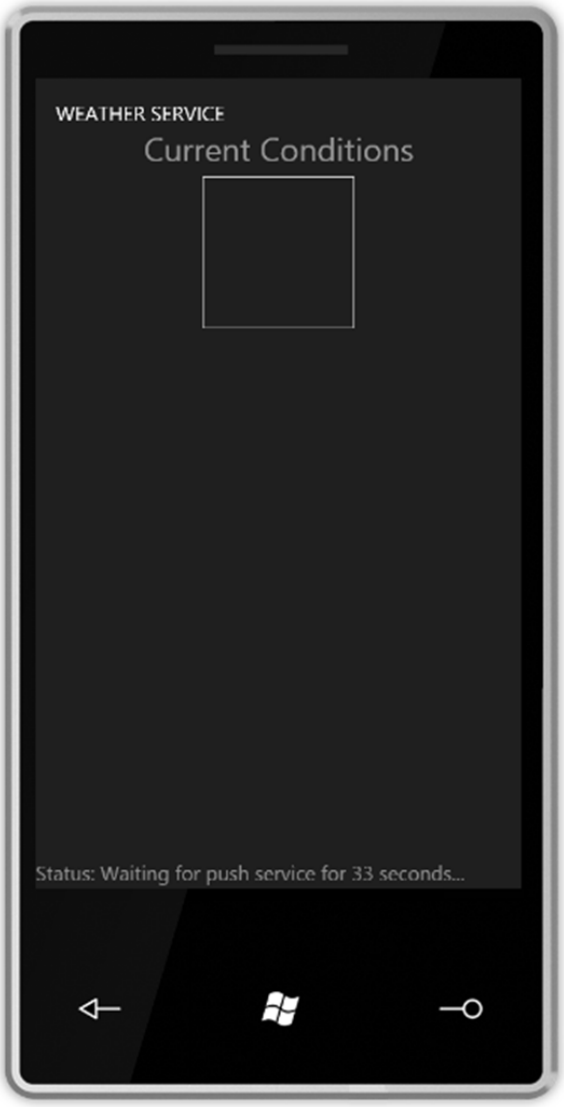


Figure 24

L'émulateur Windows Phone s'exécutant pour la première fois

19. Arrêtez le débogage (***ne fermez pas la fenêtre de l'émulateur !***) et exécutez à nouveau l'application.

20. Assurez-vous que l'application Windows Phone 7 s'est correctement enregistrée auprès du service :

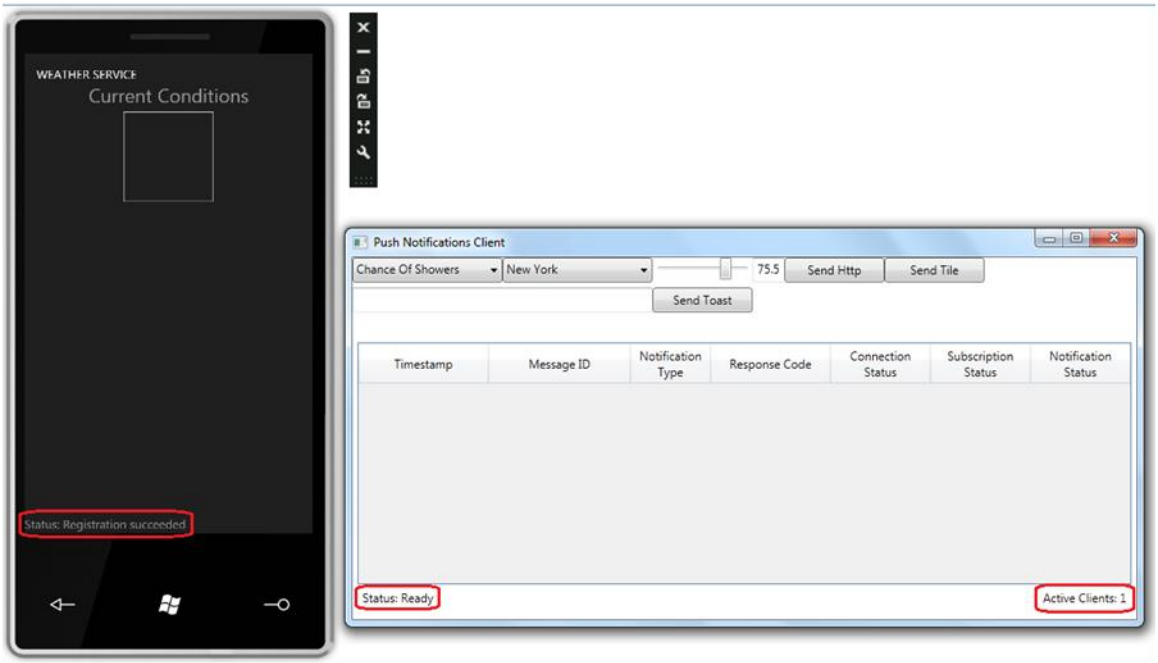


Figure 25

Vérification de l'enregistrement auprès du service

21. Mettez un point d'arrêt sur la méthode **httpChannel_HttpNotificationReceived**

```
void httpChannel_HttpNotificationReceived(object sender, HttpNotificationEventArgs e)
{
  Trace("================================================");
  Trace("RAW notification arrived:");
```

Figure 26

Point d'arrêt dans le gestionnaire de notification

22. Changez quelques paramètres de l'application WPF et cliquez sur **Send Http**. Lorsque le point d'arrêt est rencontré et que l'exécution de code est suspendue, examinez les informations reçues.

```
void httpChannel_HttpNotificationReceived(object sender, HttpNotificationEventArgs e)
{
  Trace("================================================");
  Trace("RAW notification arrived:");
```

Figure 27

Point d'arrêt rencontré lorsque la notification arrive

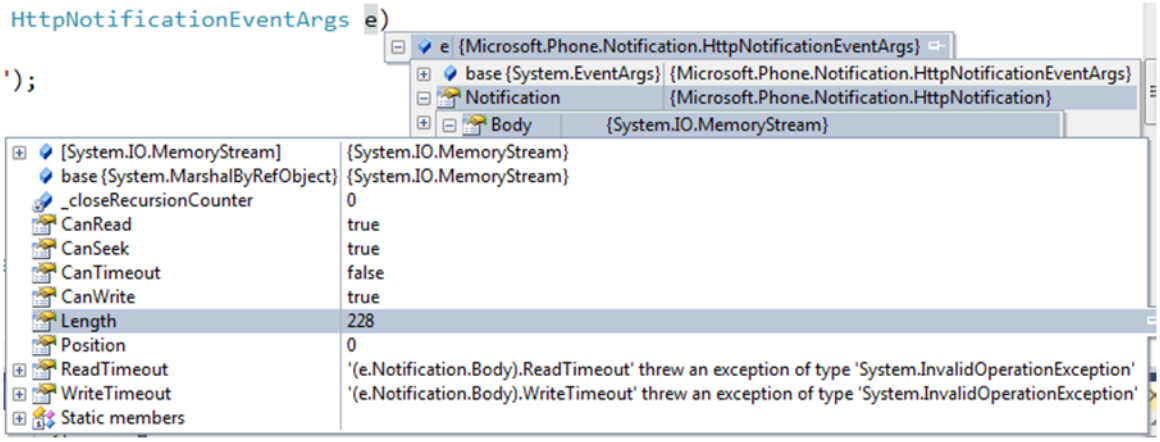


Figure 28

Notification des données reçues par l'application Windows Phone 7

23. Appuyez sur **F5** pour continuer l'exécution du programme. Observez la nouvelle entrée dans le log de l'application cliente WPF.

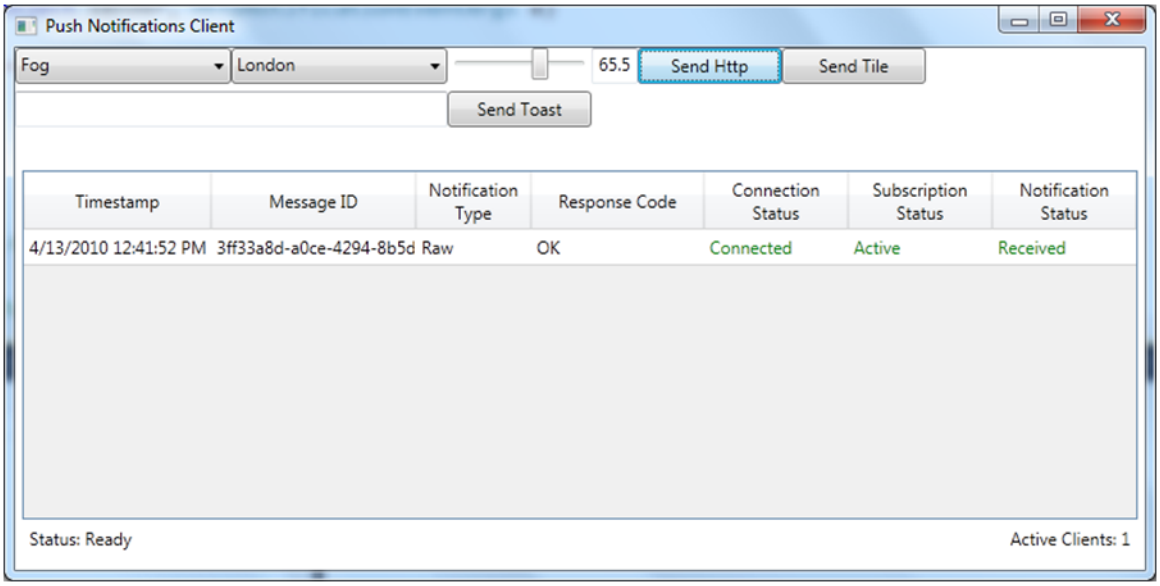


Figure 29

Mise à jour du log de l'application cliente WPF

24. Arrêtez le débogage et retournez dans Visual Studio (***ne fermez pas la fenêtre de l'émulateur !***)

Etape 4 – Recevoir et traiter les évènements du service de notification

1. Dans cette étape, vous allez créer une méthode qui parse le XML qui arrive et qui met à jour l'interface graphique de l'application Windows Phone 7. Ajoutez cette méthode dans la région **Misc logic** :

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs –ParseRAWPayload function*)

C#

```
private void ParseRAWPayload(Stream e, out string weather,        out
string location, out string temperature)
{
   XDocument document;

   using (var reader = new StreamReader(e))
   {
      string payload = reader.ReadToEnd().Replace('\0',
             ' ');
      document = XDocument.Parse(payload);
   }

   location = (from c in  document.Descendants("WeatherUpdate")
       select c.Element("Location").Value).FirstOrDefault();
   Trace("Got location: " + location);

   temperature = (from c in      document.Descendants("WeatherUpdate")
     select  c.Element("Temperature").Value).FirstOrDefault();
   Trace("Got temperature: " + temperature);

   weather = (from c in   document.Descendants("WeatherUpdate")
     select  c.Element("WeatherType").Value).FirstOrDefault();
}
```

2. Afin de représenter de façon graphique le temps, vous allez rajoutez des icônes représentant les conditions météo. Dans le projet **PushNotifications**, créez un répertoire **Images**.

3. Ajoutez toutes les images du répertoire **Assets** (situé à *{LAB_PATH}\Source\Assets):*

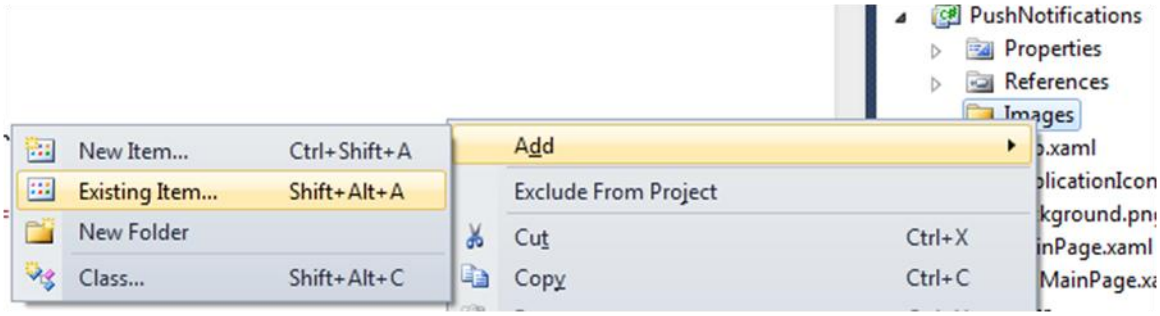


Figure 30

Ajout des images existantes du répertoire Assets

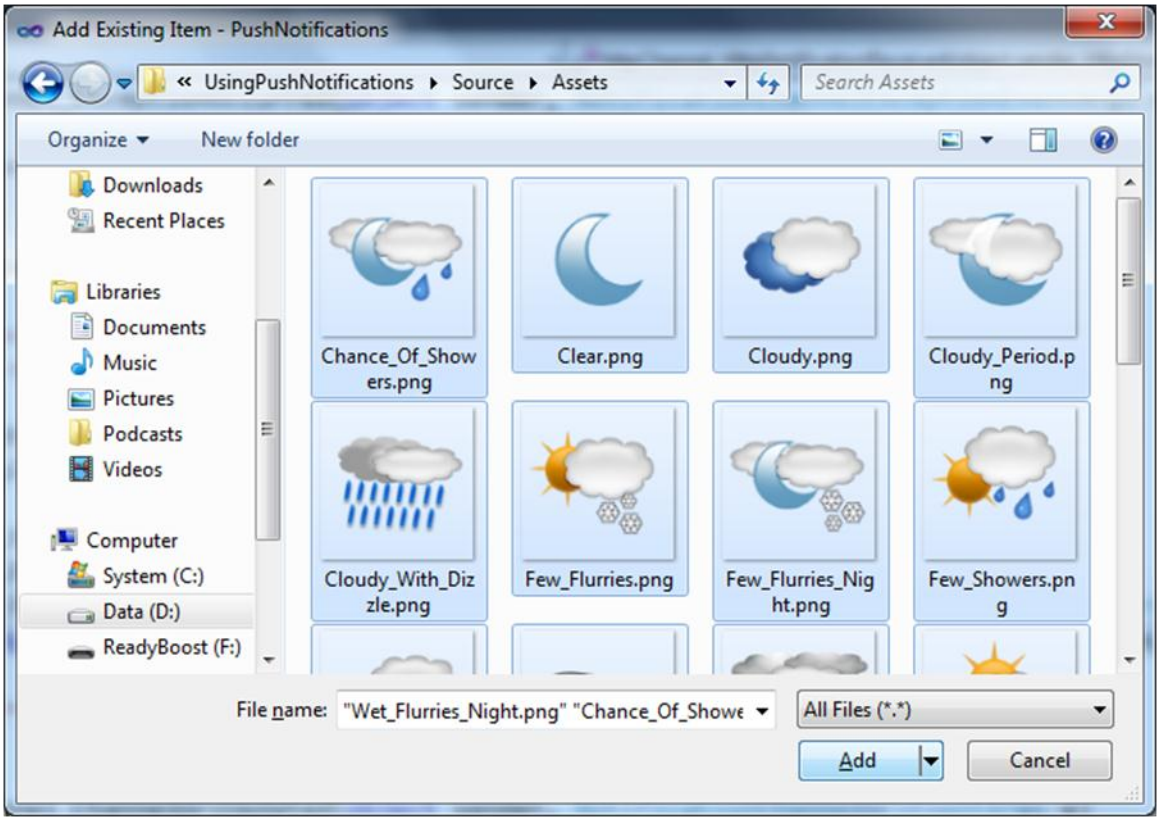


Figure 31

Ajout des images existantes du répertoire Assets

4. Marquez toutes ces images comme contenu dans leur action de génération. Pour cela, ouvrez une image, allez dans ses propriétés et sélectionnez **Build Action** | **Content**.

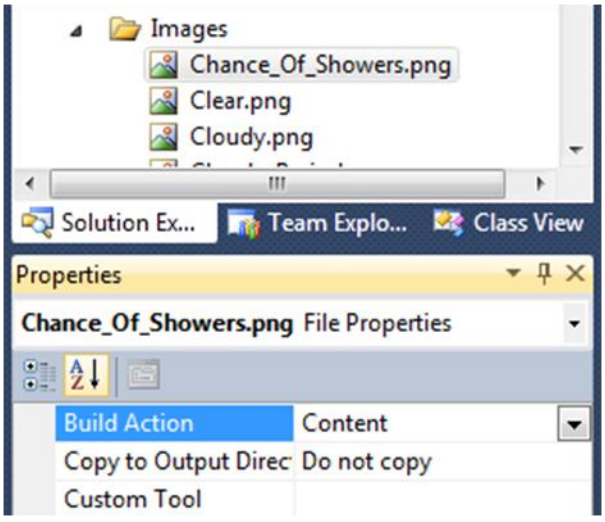


Figure 32

Marquage d'une image comme contenu

5. Vous souvenez-vous de la dernière méthode précédemment créée (**httpChannel_HttpNotificationReceived**) ? Cette méthode appelle la fonction de parsing créée précédemment et met à jour l'interface graphique. Ajoutez ce code après "*//TODO - add parsing and UI updating logic here*":

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs – httpChannel_HttpNotificationReceived function body*)

C#

```
string weather, location, temperature;
ParseRAWPayload(e.Notification.Body, out weather, out      location, out
temperature);

Dispatcher.BeginInvoke(() => this.textBlockListTitle.Text =
      location);
Dispatcher.BeginInvoke(() => this.txtTemperature.Text =    temperature);
Dispatcher.BeginInvoke(() =>     this.imgWeatherConditions.Source = new
BitmapImage(new     Uri(@"Images/" + weather + ".png",
      UriKind.Relative)));
Trace(string.Format("Got weather: {0} with {1}F at location {2}",
weather, temperature, location));
```

6. Compilez et exécutez l'application. Après son démarrage, changez des valeurs dans l'application WPF et cliquez sur **Send Http**. Observez comment la notification arrive au client Windows Phone 7 et comment l'interface graphique est mise à jour. Observez également les traces dans la fenêtre **Output** de Visual Studio 2010 (si la fenêtre n'est pas affichée, cliquez sur **Debug** | **Windows** | **Output** ou Ctrl+W, O).

Figure 33

Notification Http parsée

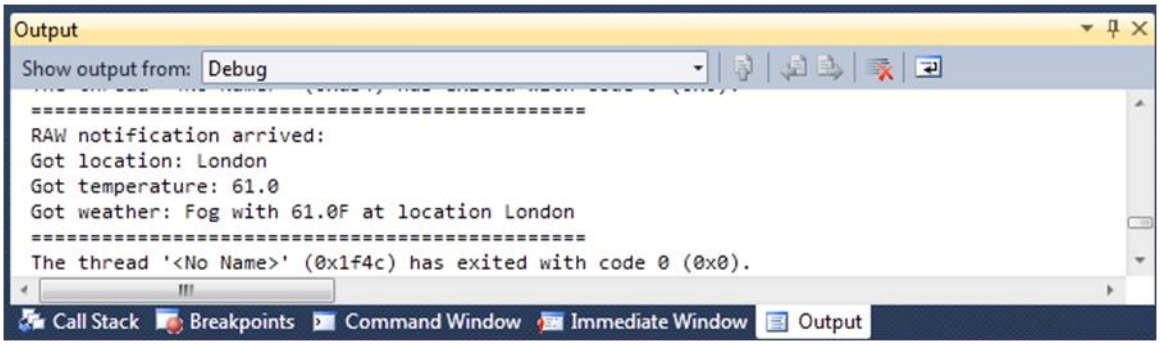


Figure 34

Information de trace dans la fenêtre Output

7. Envoyez différentes notification et observez l'interface graphique changer.

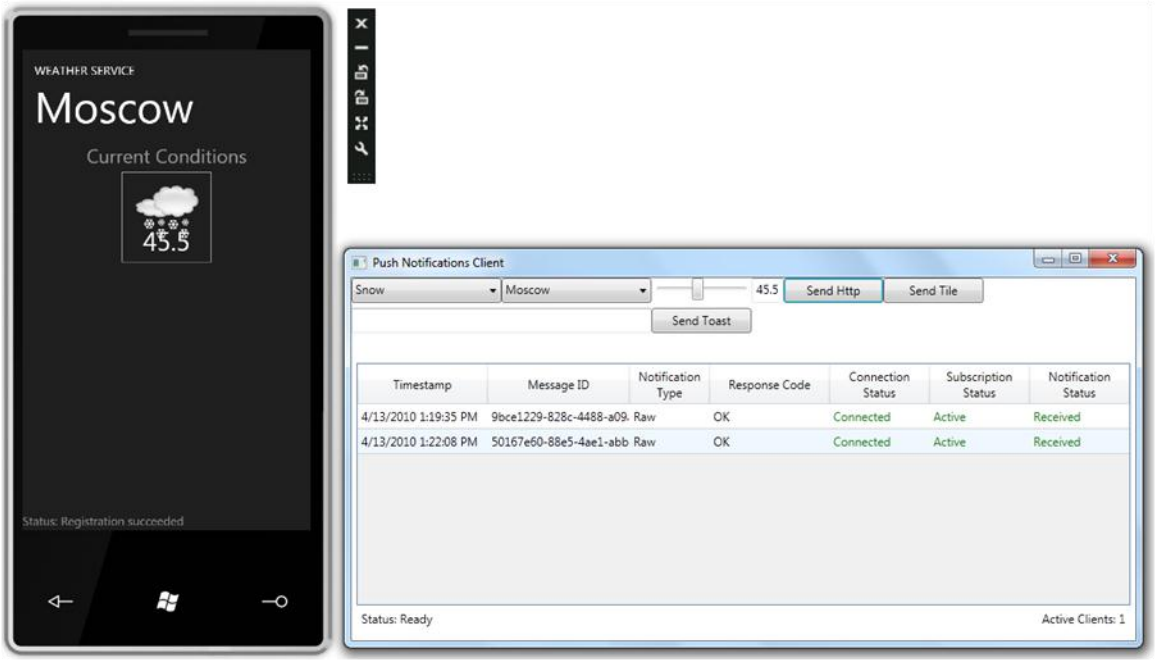


Figure 35

Notification Http parsée. Log mis à jour.

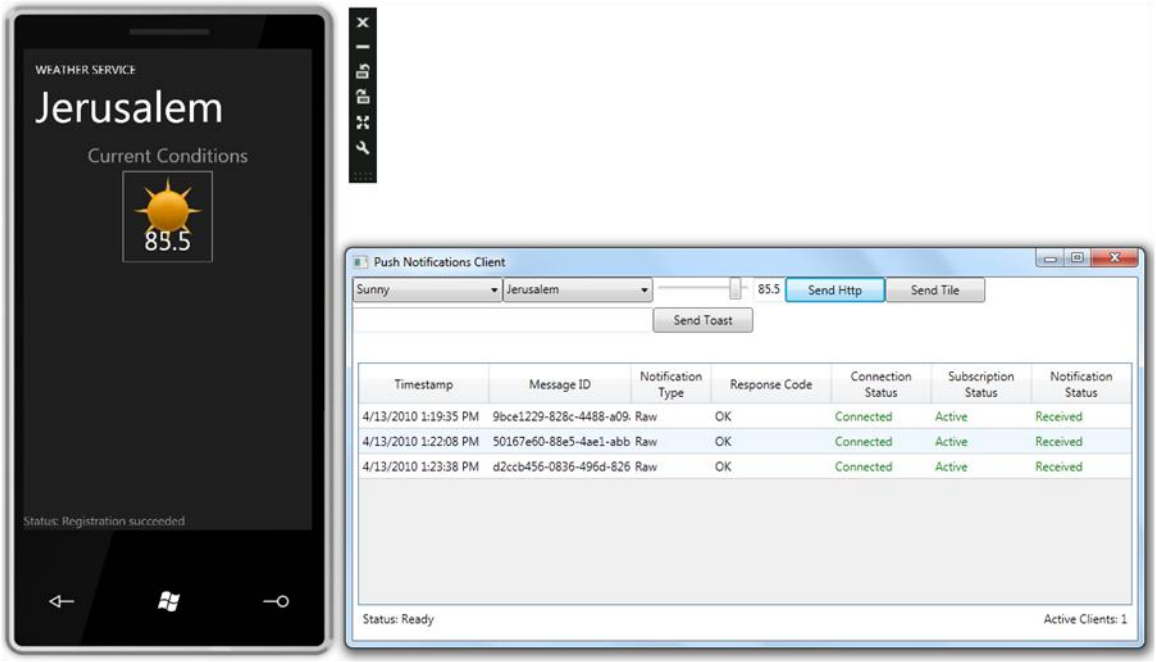


Figure 36

Notification Http parsée. Log mis à jour.

8. Arrêtez le débogage et retournez dans le code.

Comme mentionnée précédemment, l'application doit essayer de restaurer un canal MPN avant d'en créer un nouveau. Pour cela, vous devez appeler la méthode Find en espérant que cela vous renvoie un HttpChannelNotification

9. Ajoutez ce bout de code, qui présente 2 fonctions pour sauver et charger les informations de canal dans l'Isolated Storage d'une application Windows Phone 7 (pour plus d'information sur l'Isolated Storage, rendez-vous ici : http://www.silverlight.net/learn/quickstarts/isolatedstorage/) :

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs –Saving and Loading Channel information functionality*)

C#

```
#region Loading/Saving Channel Info
private bool TryFindChannel()
{
   bool bRes = false;

   Trace("Getting IsolatedStorage for current Application");
   using (IsolatedStorageFile isf =
       IsolatedStorageFile.GetUserStoreForApplication())
   {
      Trace("Checking channel data");
      if (isf.FileExists(fileName))
      {
         Trace("Channel data exists! Loading...");
         using (IsolatedStorageFileStream isfs = new
       IsolatedStorageFileStream(fileName,
       FileMode.Open, isf))
         {
            using (StreamReader sr = new StreamReader(isfs))
            {
               string uri = sr.ReadLine();
               Trace("Finding channel");
               httpChannel =
             HttpNotificationChannel.Find(channelName);

               if (null != httpChannel)
               {
                  if (httpChannel.ChannelUri.ToString() ==
                    uri)
                  {
                     Dispatcher.BeginInvoke(() =>
             UpdateStatus("Channel retrieved"));
                     SubscribeToChannelEvents();

                     SubscribeToService();
                     bRes = true;
```

```
                }
                sr.Close();
            }
        }
    }
    else
        Trace("Channel data not found");
}

return bRes;
}

private void SaveChannelInfo()
{
Trace("Getting IsolatedStorage for current Application");
using (IsolatedStorageFile isf =
    IsolatedStorageFile.GetUserStoreForApplication())
{
    Trace("Creating data file");
    using (IsolatedStorageFileStream isfs = new
     IsolatedStorageFileStream(fileName,
     FileMode.Create, isf))
    {
        using (StreamWriter sw = new StreamWriter(isfs))
        {
            Trace("Saving channel data...");
            sw.WriteLine(httpChannel.ChannelUri.ToString());
            sw.Close();
            Trace("Saving done");
        }
    }
}
}
#endregion
```

10. Changez le constructeur de la classe pour appeler la fonction de chargement écrite juste avant :

C#

```
public MainPage()
{
    InitializeComponent();

    SupportedOrientations = SupportedPageOrientation.Portrait      |
SupportedPageOrientation.Landscape;

    if (!TryFindChannel())
        DoConnect();
```

```
}
```

11. Ajoutez un appel à la méthode de sauvegarde du canal dans la fonction **httpChannel_ChannelUriUpdate**. Ajoutez ce code tout de suite après “Channel opened” :

(Code Snippet – *Using Push Notifications – MainPage.xaml.cs –SaveChannelInfo function call*)

C#

```
void httpChannel_ChannelUriUpdated(object sender,
NotificationChannelUriEventArgs e)
{
    Trace("Channel opened. Got Uri:\n" +
httpChannel.ChannelUri.ToString());
    Dispatcher.BeginInvoke(() => SaveChannelInfo());

    Trace("Subscribing to channel events");
    SubscribeToService();

    Dispatcher.BeginInvoke(() => UpdateStatus("Channel created
successfully"));
}
```

12. Compilez et exécutez l’application deux fois. Utilisez le débuggeur pour vérifier que la première fois, l’application ouvre un canal et sauvegarde ces informations, et que la seconde fois, retrouve le canal par son nom.

    Cette étape conclut l’exercice. Durant cet exercice, vous avez appris comment communiquer avec le service de notification de type Push de Microsoft, comment préparer et envoyer des données à un client Windows Phone 7 et comment recevoir des messages.

> **Note :** La solution complète de l’exercice se trouve dans le répertoire : \Source\Ex1-RawNotifications\End

Résumé

Dans cet atelier, vous avez appris tout ce qu’il faut savoir sur le service de notification de type Push pour la plateforme Windows Phone 7. Vous avez découvert les types de notification ainsi que comment préparer et envoyer des données.