Hands-On Lab

Using Push Notifications

CONTENTS

Overview

The Microsoft Push Notification Service in Windows Phone offers third party developers a resilient, dedicated, and persistent channel to send information and updates to a mobile application from a web service.

In the past, a mobile application would need to frequently poll its corresponding web service to know if there are any pending notifications. While effective, polling results in the device radio being frequently turned on, impacting battery life in a negative way. By using push notifications instead of polling, a web service can notify an application of important updates on an as-needed basis.

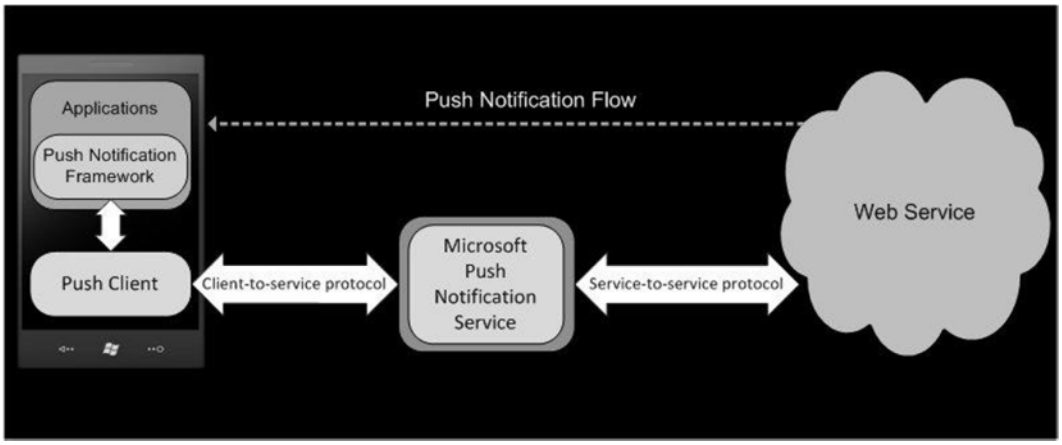


Figure 1

Push Notifications

When a web service has information to send to an application, it sends a push notification to the Push Notification Service, which in turn routes the push notification to the application. Depending on the format of the push notification and the payload attached to it, the information is delivered as raw data to the application, the application's tile is visually updated, or a toast notification is displayed. The application can then contact the web service using its own protocol, if needed.

The Push Notification Service sends a response code to your web service after a push notification is sent. However, the Push Notification Service does not provide an end-to-end confirmation that your push notification was delivered from your web service to your application. For more information, see Push Notification Service Response Codes for Windows Phone

This lab covers the push notification and also introduces the usage of http services in Silverlight. During this lab you will create server side logic needed to send messages

through Push Notification Service. You will create a simple Windows Phone 7 application which serves as a client to receive such notifications. The client application will receive weather updates. The server side business application (simple WPF application) will send weather alerts to registered client applications through Push Notification Services. Once client Windows Phone 7 application will receive such alert it will display received information.

Objectives

At the end of the lab you will:

- Familiarize with the communication capabilities of Windows Phone 7 application
- Familiarize with the push notification concepts and the behaviors they enable on the phone
- Understand how push notification works on the phone and in the cloud
- Use the phone push notification services to create a subscription for Tokens (tiles), Toasts, and raw push notification
- Use web client to register for Push Notifications
- Use the network status to display the current status of the phone network
- Create a SL application that register for push notification services (both token and toast)
    - Handle push events (token, toast, and raw)during run time
    - Show token and toast on shell

Prerequisites

The following is required to complete this hands-on lab:

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

> **Note:** All of these Tools can be downloaded together in a single package from http://developer.windowsphone.com

Setup

For convenience, much of the code used in this hands-on lab is available as Visual Studio code snippets. To install the code snippets:

1. Run the .**vsi** installer located in the lab's **Source\Setup** folder.

> **Note:** If you have issues running the code snippets installer you can install the code snippets manually by copying all the .snippet files located in the **Source\Setup\CodeSnippets** folder of the lab to the following folder:
>
> **My Documents\Visual Studio 2010\Code Snippets\Visual C#\My Code Snippets**

Using the Code Snippets

With code snippets, you have all the code you need at your fingertips. The lab document will tell you exactly when you can use them. For example,

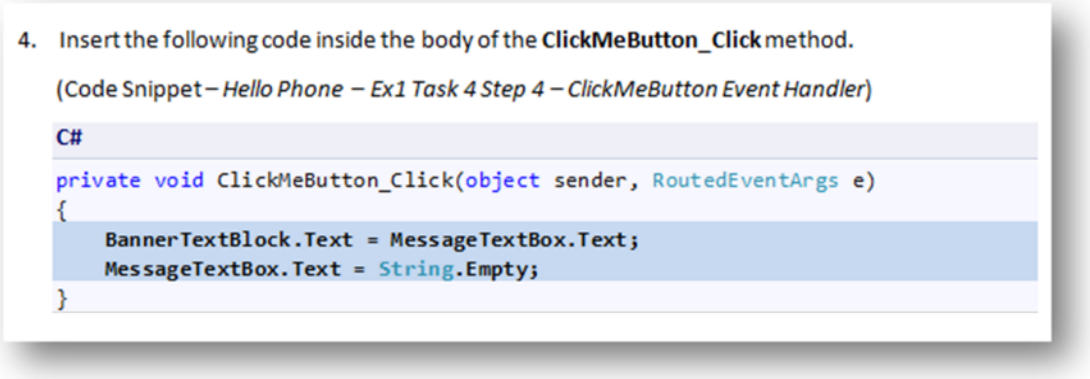


Figure 2

Using Visual Studio code snippets to insert code into your project

To add this code snippet in Visual Studio, you simply place the cursor where you would like the code to be inserted, start typing the snippet name (without spaces or hyphens), watch as IntelliSense picks up the snippet name, and then press the Tab key twice when the snippet you want is selected. The code will be inserted at the cursor location.

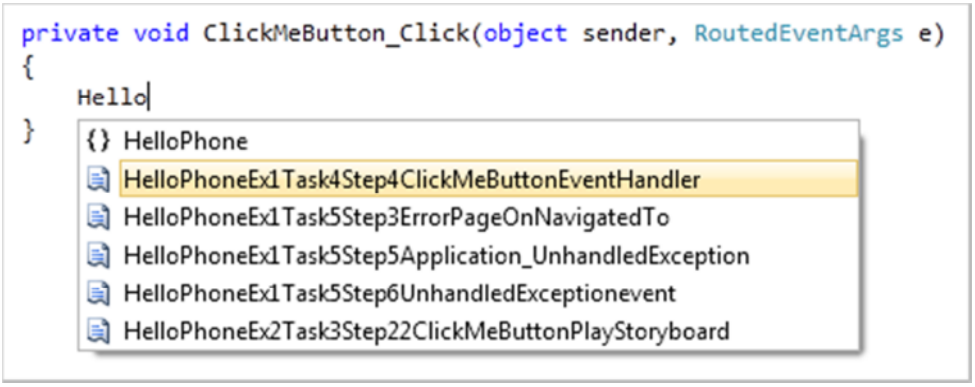


Figure 3

Start typing the snippet name

Figure 4

Press Tab to select the highlighted snippet

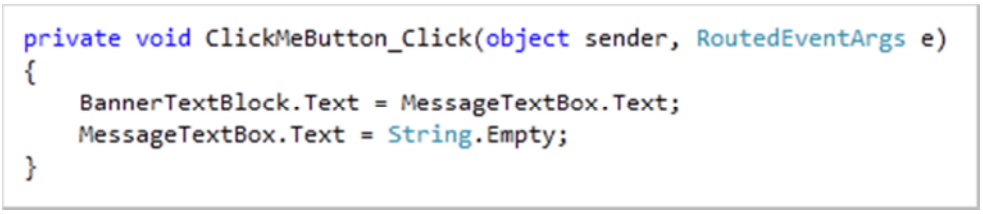


Figure 5

Press Tab again to expand the snippet

To insert a code snippet using the mouse rather than the keyboard, right-click where you want to insert the code snippet, select **Insert Snippet** followed by **My Code Snippets** and then pick the relevant snippet from the list.

To learn more about Visual Studio IntelliSense Code Snippets, including how to create your own, see http://msdn.microsoft.com/en-us/library/ms165392.aspx.

Exercises

This hands-on lab comprises the following exercises:

1. Introduction to the Windows Phone RAW notifications for Updates
2. Introduction to the Toast and Tile Notifications for Alerts

Estimated time to complete this lab: **90 minutes**.

Exercise 1: Introduction to the Windows Phone RAW Notifications for Updates

In this section we will open the starter solution and:

- Implement server side notification and registration services
- Create Windows Phone 7 client application
- Create notification channel and subscribe to channel events
- Receive and process events from Push Notification Services

We will use the Microsoft Visual Studio 2010 Express for Windows Phone development environment, and will deploy to the Windows Phone Emulator for debugging. The solution we will be working with is based upon the Silverlight for Windows Phone Application template. During development, we will add a Silverlight for Windows Phone project specific item, the Windows Phone Portrait Page.

> **Note:** The steps in this hands-on lab illustrate procedures using Microsoft Visual Studio 2010 Express for Windows Phone, but they are equally applicable to Microsoft Visual Studio 2010 with the Windows Phone Developer Tools. Instructions that refer generically to Visual Studio apply to both products.

Task 1 – Creating the Weather Service Solution

In this task, you use a provided starter solution for Microsoft Visual Studio 2010 Express for Windows Phone or Microsoft Visual Studio 2010. This solution includes the simple WPF client application which will be used send messages to the Windows Phone 7 application via Microsoft Push Notification Service and will host WCF registration service. This service will be created during this task. The provided WPF application will self-host REST WCF service. For this purpose the project already has all requested configurations.

1. Open Microsoft Visual Studio 2010 Express for Windows Phone from **Start | All Programs | Microsoft Visual Studio 2010 Express | Microsoft Visual Studio 2010 Express for Windows Phone**.

   > **Visual Studio 2010**: Open Visual Studio 2010 from **Start | All Programs | Microsoft Visual Studio 2010**.

Important note: In order to run self-hosted WCF services within Visual Phone 2010 Express for Windows Phone or Microsoft Visual Studio 2010 it should be opened in Administrative Mode. For reference about creating and hosting self-hosted WCF services see MSDN article (http://msdn.microsoft.com/en-us/library/ms731758.aspx). In order to open Visual Studio 2010 Express for Windows Phone or Visual Studio 2010 in Administrative Mode locate the Microsoft Visual Studio 2010 Express for Windows Phone shortcut at **Start | All Programs | Microsoft Visual Studio 2010 Express** or Microsoft Visual Studio 2010 shortcut at **Start | All Programs | Microsoft Visual Studio 2010**, right-click on the icon and select "Run as administrator" from the opened context menu. The UAC notification will pop up. Click "Yes" in order to allow running the Visual Studio 2010 Express for Windows Phone or Visual Studio 2010 with elevated permissions.

2. In the **File** menu, choose **Open Project**.

Visual Studio 2010: In the File menu, point to **open** and then select **Project/Solution**.

3. Navigate to the starter project location at the **Source\Ex1-RawNotifications\Begin** folder of this lab, select **Begin.sln** and click **Open**.

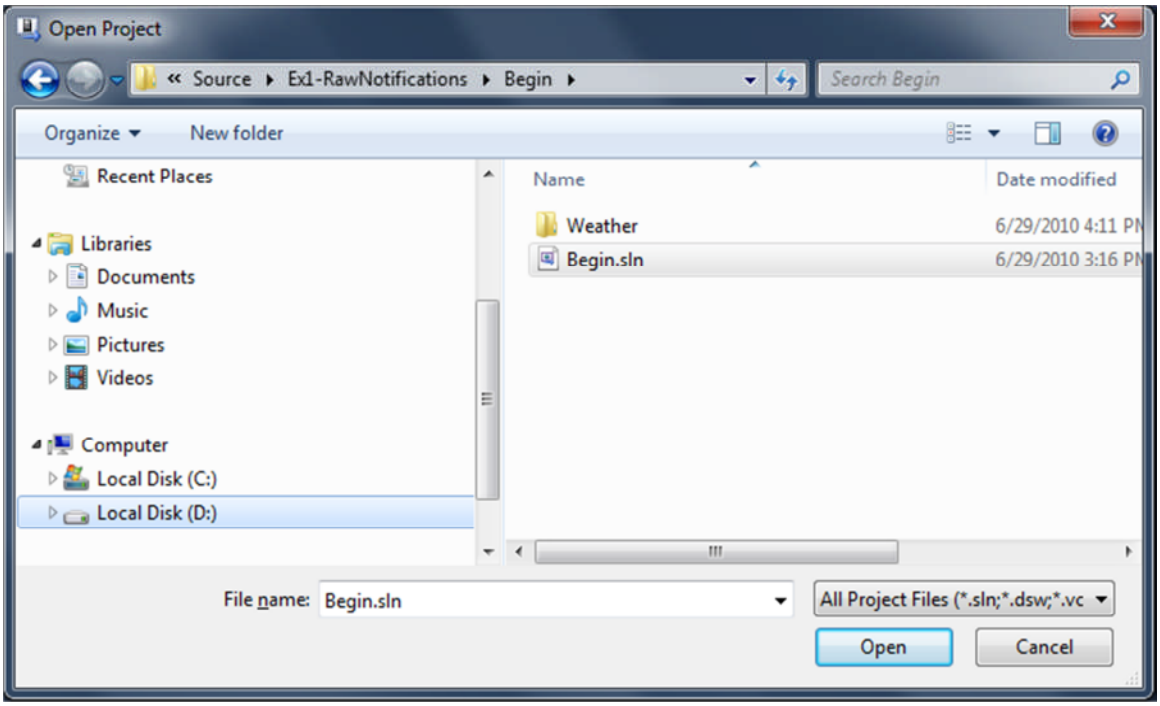


Figure 6

Opening starter project

4. Examine opened project:
    a. This is standard WPF application.

    > **Note:** This application targets .NET 4 Framework and not .NET 4 Framework Client Profile in order to support self-hosted RESTful WCF service.

    b. The WPF application consists from MainWindow screen, PushNotificationsLogViewer user control and StatusToBrush value converter.
    c. In “Service” project folder included interface definition for WCF RESTfull service.
    d. Aside standard WPF application references, this application references also System.ServiceModel and System.ServiceModel.Web assemblies to support RESTful WCF services.
5. Press **F5** to compile and run the application. Familiarize yourself with it and then close application and get back to the Visual Studio. The application allows you to send Tile, Toast and Raw HTTP notifications

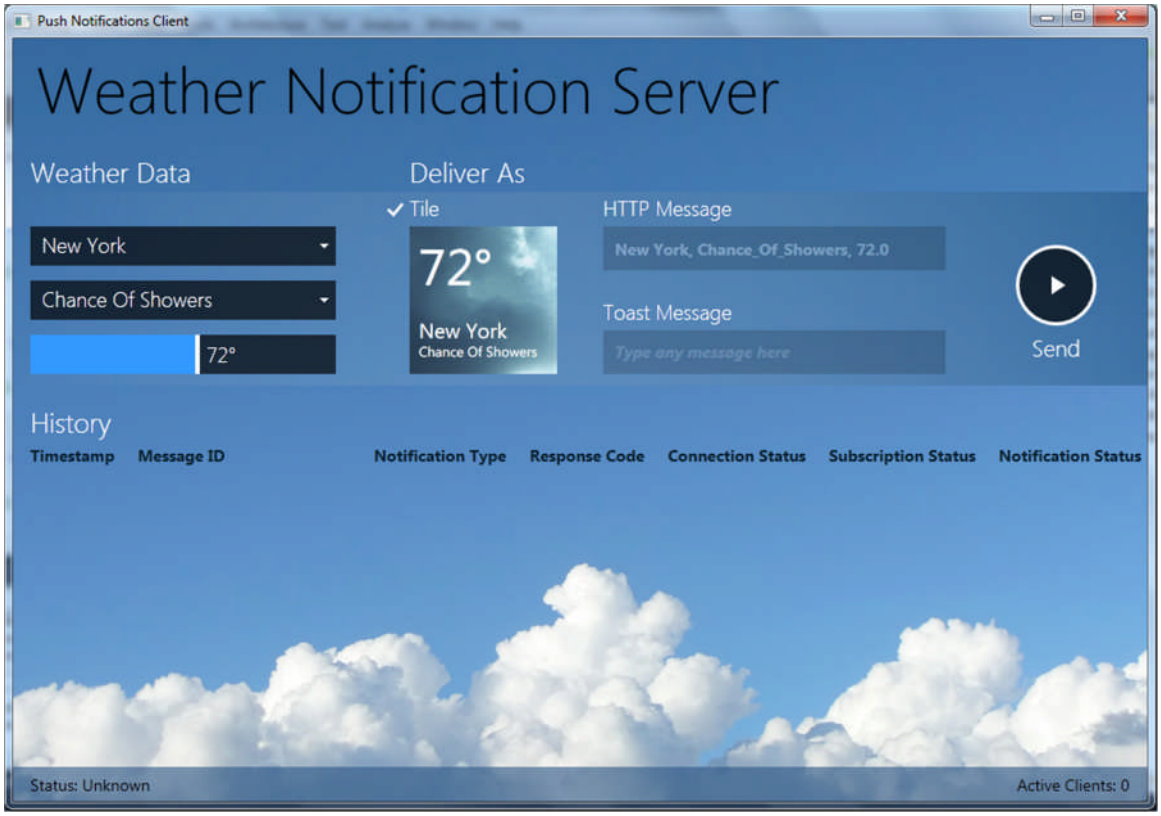


Figure 7

Push Notifications Client Application

6. During the following few steps you will create a WCF service instance and initialize it. In order to communicate with Push Notification Service the caller should provide URI of the cannel registered with the service. You will create such a channel and register it with the service in next tasks during this lab. To start, add new class to the **Service** project folder. To add a new class to the project folder, right-click on the project folder name, select **Add** and then **Class**.

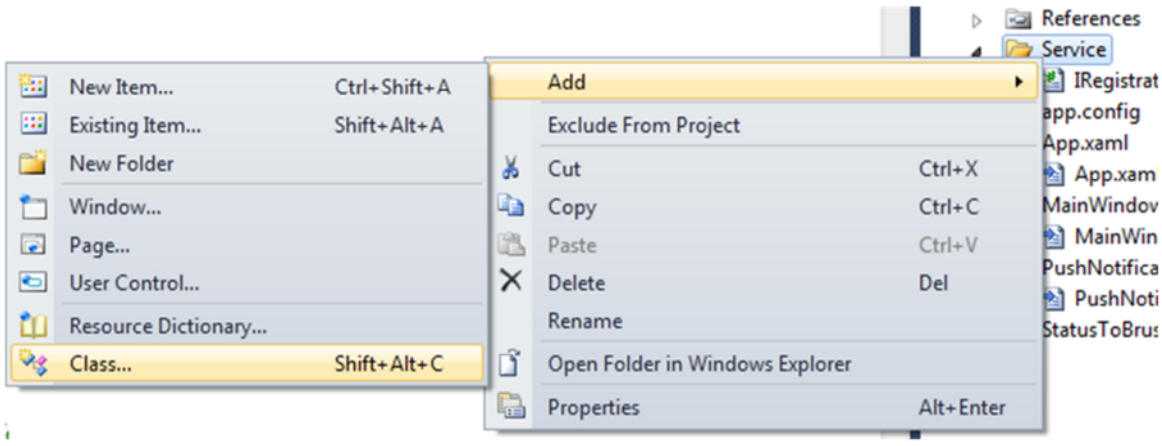


Figure 8

Adding New Class to the Project

7. Name it **RegistrationService** and click the **Add** button.

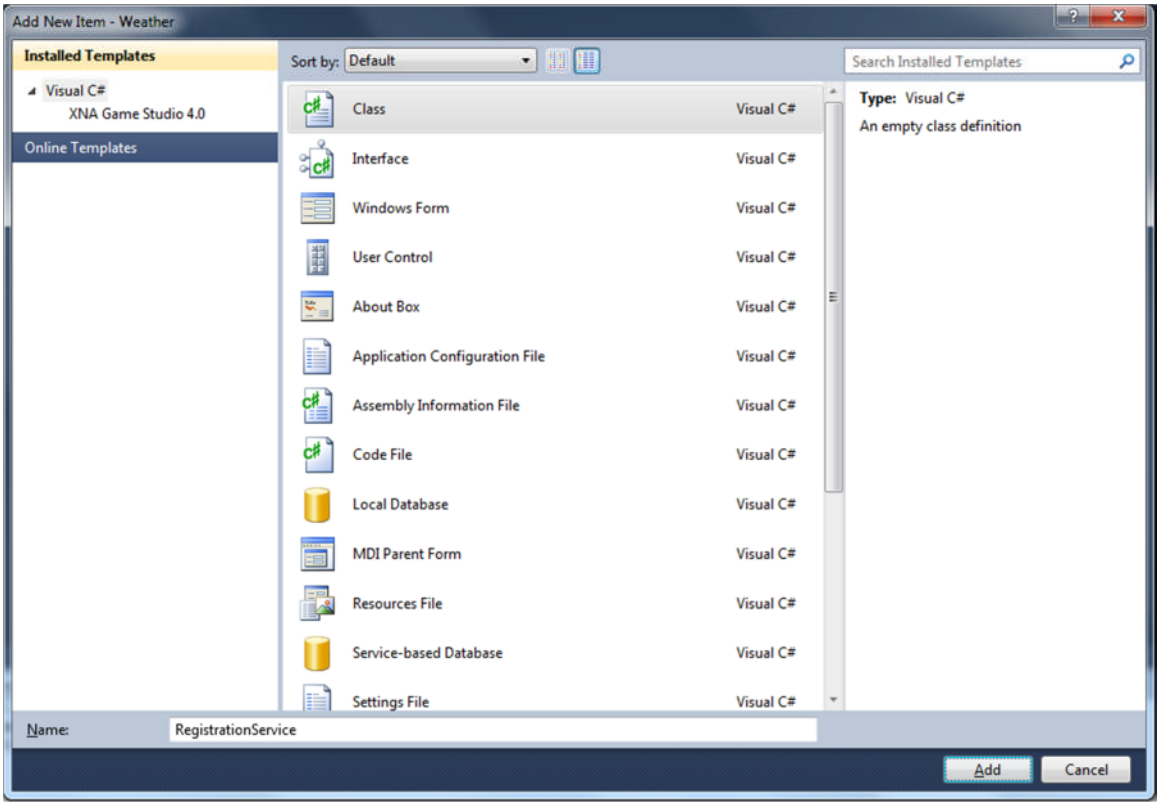


Figure 9

Name a new class

8. Open the created class if not already opened.
9. Make the created class public and implement **IRegistrationService** interface. This interface defines a set of methods that allow phone to register their URI.

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

C#

```
public class RegistrationService : IRegistrationService
{

}
```

10. Click on **IRegistrationService**, open context menu by hovering mouse over "_" sign.

Figure 10

Opening interface options

11. Select **Implement interface IRegistrationService** to implement the interface the default way.

Figure 11

Implementing interface

12. Resulting class code should looks like the following code snippet. Notice that you can add the **IRegistrationService Members** surrounding region.

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
}
```

13. The registration service holds all the registered clients in internal list, and notifies on client registration/un-registration. In addition it will check that client requested in registration not yet registered on order to avoid double registration. To support this functionality add following (public and private) class variables:

(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. Replace the **Register** function body with the following code snippet:

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

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

15. Replace the **Unregister** function body with the following code snippet:

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

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

16. Create a new region with **Subscribe** and **Unsubscribe** helper functions according to the following code snippet:

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

```
C#
#region Subscription/Unsubscribing 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. Create helper **OnSubscribed** function according to the following code snippet:

    (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. To hold Subscribed event arguments, create the **SubscriptionEventArgs** internal class (inside the **RegistrationService** class) according to the following code snippet:

    (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. Lastly create a public function in the class that returns the list of all subscribers – it will be used later by the WPF client application.

    (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. Open **App.xaml.cs** file.
21. Locate the code line with "*//TODO - remove remark after creating registration service*" and remove the remark from WCF service host initialization (as shown below):

C#

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

22. Compile and run the application. Once application started, check that the RESTful WCF service works. To do it, add a break point to the **Register** and/or **Unregister** functions in **RegistrationService.cs**.

```
public void Register(string uri)
{
    Uri channelUri = new Uri(uri, UriKind.Absolute);
    Subscribe(channelUri);
}

public void Unregister(string uri)
{
    Uri channelUri = new Uri(uri, UriKind.Absolute);
    Unsubscribe(channelUri);
}
#endregion
```

Figure 12

Breakpoint to check RESTful WCF service

23. Launch Internet Explorer and navigate to the following address:

    http://localhost:8000/RegirstatorService/Register?uri=http://www.microsoft.com

24. When breakpoint hit occurs, check that received URI address is http://www.microsoft.com.

```
public void Register(string uri)
{
    Uri channelUri = new Uri(uri, UriKind.Absolute);
    Subscribe(channelUri);
}
```

uri "http://www.microsoft.com"

Figure 13

Checking RESTful WCF service

25. Stop the debugging, remove break points and close Internet Explorer.
26. In order to communicate with the Push Notification Service you will create a utility class. This class will hold all the communication logic. Add the **NotificationSenderUtility** project from the **Source\Assets** folder of this lab. This is an empty class library project that references the .NET 4 Framework and not .NET 4 Framework Client Profile.
27. Open the **NotificationSenderUtility.cs** empty class.
28. This class holds all the functionality needed to communicate with Microsoft Push Notification Service. During next couple steps you will add the

functionality to prepare and send the message to the Push Notification Services. Along with this you will expose public functionality which will be used by WPF Client application and will enable it to send messages to the registered clients. This class will use asynchronous pattern to communicate with Push Notification Service and will notify the callers by calling them back with provided callback functions.

In order to ease up on creating business logic and managing various notification types, add a new **NotificationType** Enum to the namespace (just above or below **NotificationSenderUtility** class)

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

C#

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

29. Next, right after the **Enum** definition (outside of **NotificationSenderUtility** class) add a class which will hold the data received from Push Notification Service as a response to the HTTP post to the MSPNS:

(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_HE
ADER];
        this.DeviceConnectionStatus =
response.Headers[NotificationSenderUtility.DEVICE_CONNECTION_STAT
US_HEADER];
        this.SubscriptionStatus =
response.Headers[NotificationSenderUtility.SUBSCRIPTION_STATUS_HE
ADER];
    }

    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. Next inside the **NotificationSenderUtility** class add constants related to the Push Notification Service communication.

    (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. Create the delegate for callback function and add the helper function to execute the delegated callback:

(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
```

32. In the next few steps you will create the functionality to send RAW HTTP notification to the Push Notification Service. During these steps you will create a generic function which prepares and sends message according to the notification type. First, create a public function called **SendRawNotifiaction** to send the RAW HTTP notification to all subscribed clients. Add the following code snippet to the **NotificationSenderUtility** class:

(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. Next, add a **SendNotificationByType** function – the generic function, which will call to the specific sending functionality:

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

C#

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

34. Microsoft Push Notification Service is a cloud service, which used to push messages to the subscribed Windows Phone 7 clients. Later during this task you will create a Windows Phone 7 application and this application will open a push channel from this service to the client. The response to such subscription process (from Push Notification Service) will be channel URI – the unique identification of the client device. In order to push notifications to the device Push Notification Service expects web call (web request) on this URI with the payload of data to send to the device. After such request been received, the Push Notification Service will locate the device and send the payload. In previous steps you created functionality needed to place a call to the service.

    The **SendMessage** function which will do all the communication will check the length of the payload and will throw exception in case it is too long. If the payload is ok, then it will use **HttpWebRequest** class (from System.Net namespace, more info: http://msdn.microsoft.com/en-

us/library/system.net.httpwebrequest.aspx) and will prepare and write the request stream to the channel URI. After sending the information the function waits for response from Push Notification Service and then notifies the caller function by calling back delegated method.

Add the following function, which will send the request to the Push Notification Service:

(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. In the try block locate "// TODO : send message to MPNS" statement and replace it with the following **HttpWebRequest** initializations:

(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";
```

36. Now, when the request is ready you need to open it asynchronously. When the stream will be opened you'll get the Stream object (from System.IO namespace, more info: http://msdn.microsoft.com/en-us/library/system.io.stream.aspx). This Stream objects represents the request stream and will be used to write (asynchronously) the payload. After finishing with writing procedure the function will switch to getting the Push Notification Service's response, and when it arrives it will notify the caller by invoking delegated function. Add the following code snippet to the function body, after the previous one (and within the *try* statement) :

    (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. Save and close the **NotificationSenderUtility** class.
38. From the **Weather** application add a reference to the **NotificationSenderUtility** library by right-clicking the **References** folder in the **Weather** project and selecting **Add Reference**.

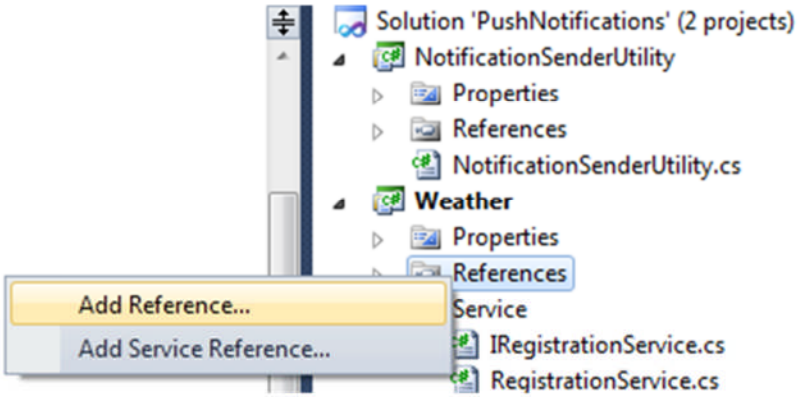


Figure 14
Adding Reference

39. In the opened dialog box click the **Projects** tab, select **NotificationSenderUtility** and click OK.

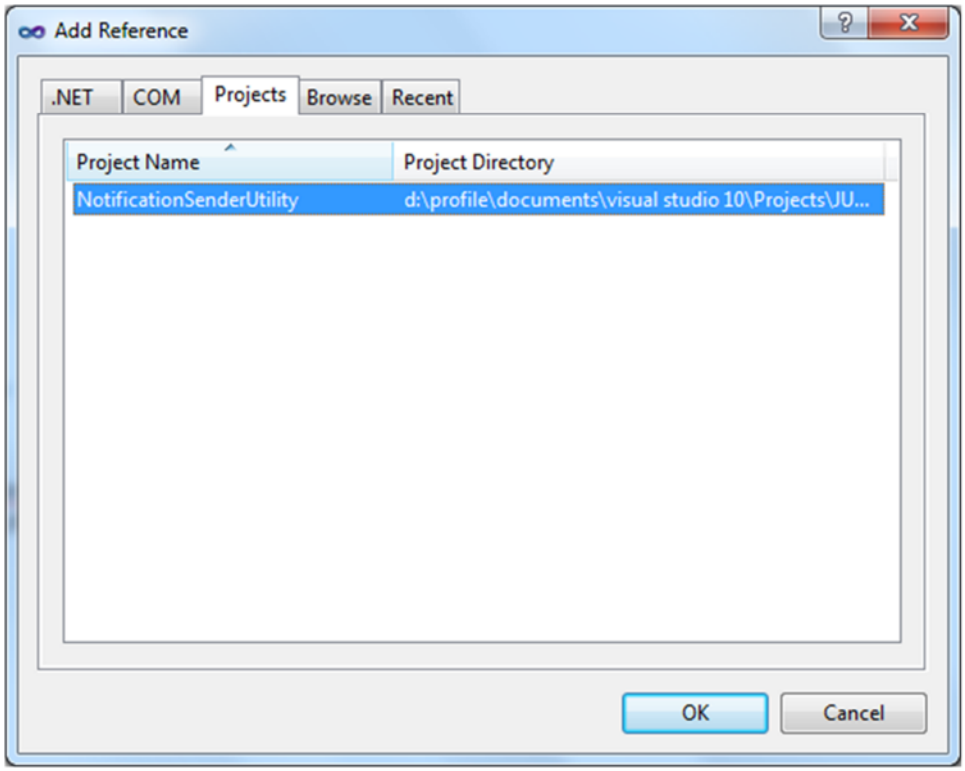


Figure 15

Adding project reference

40. Open **MainWindow.xaml.cs** located under the **Weather** project.
41. Add the following using statements to the class:

    (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. Locate the **Private variables** region and replace the "TODO" comment with the following code snippet:

    (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;
```

43. Our Windows Phone 7 client application will soon have the logic to subscribe with the previously created *Registration Service*. The WPF client application mimics real-world web site solution by pushing Weather information to the connected clients, thus it should know about client Windows Phone 7 applications and receive events on the registrations. In addition, our WPF client shows Push notification Service communication log. Locate the **MainWindow** class construction and add the following code snippet after the "TODO" comment:

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

```
C#
Log.ItemsSource = trace;
RegistrationService.Subscribed += new
EventHandler<RegistrationService.SubscriptionEventArgs>(Registrat
ionService_Subscribed);
```

44. Locate the **Event Handlers** regions and add the following function:

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

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

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

45. After the previous function, add the following function which serves as a callback function for the **NotificationSenderUtility** calls:

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

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

46. Locate the **Private functionality** region and add the following function (this function updates the WPF client application's UI):

    (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...";
}
```

47. Locate the **sendHttp** function and add the following code. This function will get the list of connected clients, prepare the payload from UI inputs and send it via NotificationSenderUtility asynchronously using a ThreadPool (from System.Threading namespace. For more info about ThreadPool refer: http://msdn.microsoft.com/en-us/library/system.threading.threadpool.aspx):

    (Code Snippet – *Using Push Notifications – MainWindow.xaml.cs – sendHttp 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: In our specific case, when the NotificationSenderUtility already implements the asynchronous pattern of communication it is not absolutely necessary to invoke the functionality via ThreadPool (asynchronously), but as a general recommendation it is best practice to invoke communication functionality such way. The communication is a long-running process, and if it will not support the asynchronous operation internally the simple blocking call to it could "freeze" the UI. That's why as a general practice use asynchronous pattern for calling potentially long-running processes.

48. Locate **Private functionality** region and add the **prepareRAWPayload** function. This function creates an XML document in-memory and returns it as a byte array (ready to be sent to the Windows Phone Push Notification Services and from there to the Windows Phone application clients). Later during this exercise you will add the functionality to parse this XML back on the device. Add the following code snippet:

(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. Locate the **RegistrationService_Subscribed** function, and add the following code snippet which resends the RAW message to the connected client before the **UpdateStatus** call:

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

C#

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

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

50. Compile the application and fix compilation errors (if any). This step concludes the task.

Task 2 – Creating the Windows Phone 7 Client Application

1. Add new project to the solution. It should be a Windows Phone Application, named **PushNotifications**.

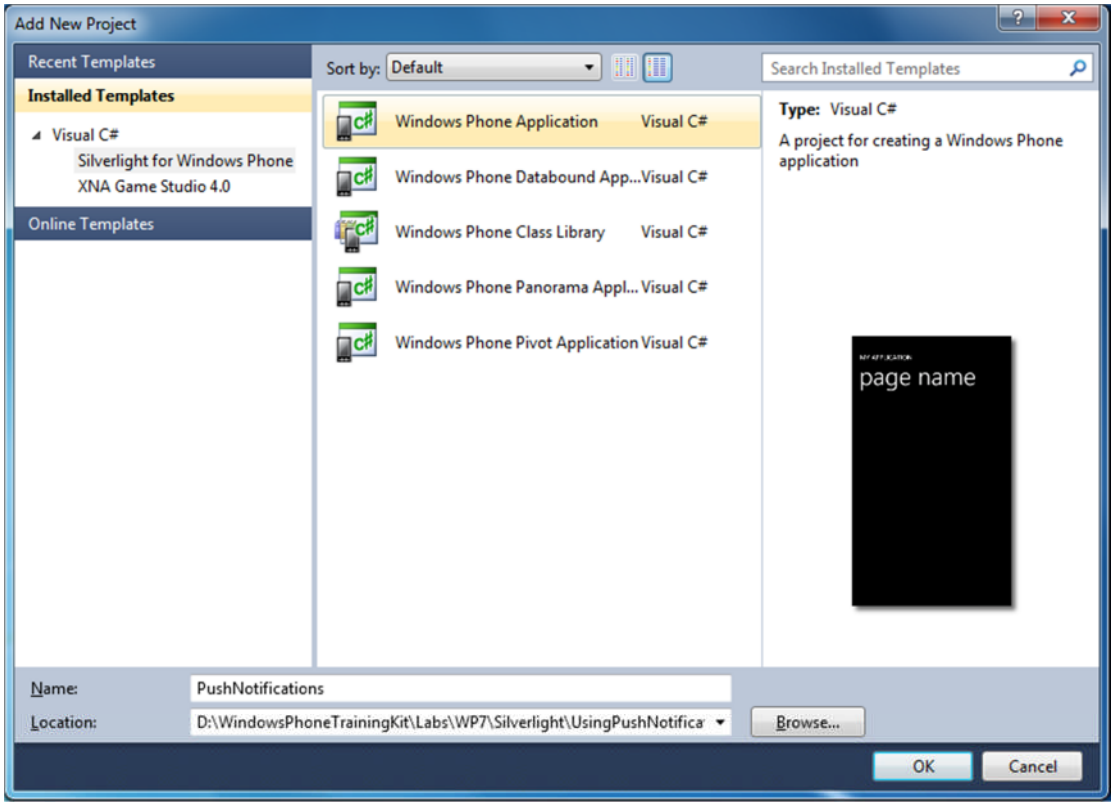


Figure 16

Adding new Windows Phone Application project to the solution

2. Add reference to the **System.Xml.Linq** assembly (from .NET tab).

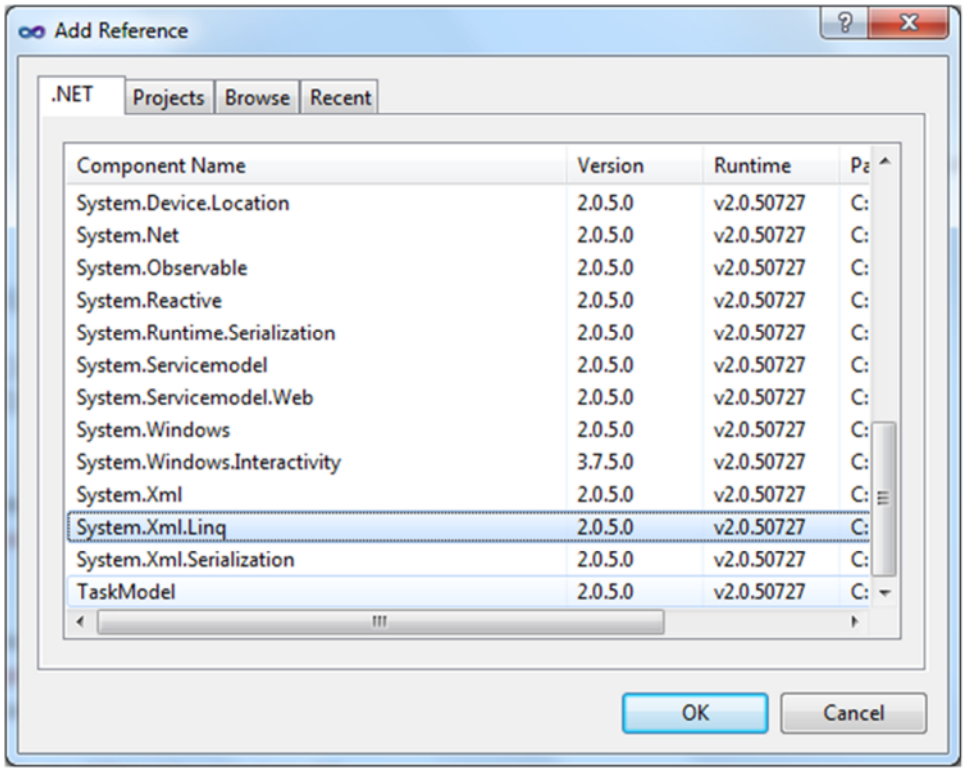


Figure 17

Adding reference to the System.Xml.Linq

3. Navigate to the **Assets** folder of this lab located in **Source\Assets** and locate file Styles.txt. Open it in Notepad.
4. Open the App.xaml.cs and copy all XAML from Styles.txt into **Application.Resources** section.
5. Add the following blue-highlighted **clr-namespace** declaration for the System CLR namespace within the Application tag (located at the top of the file).

XAML

```
<Application
    ...
    xmlns:system="clr-namespace:System;assembly=mscorlib">
    ...
</Application>
```

6. Add existing image from Assets folder named **CloudBackgroundMobile**.jpg to the project. To do it right click on **PushNotifications** (project name) and select **Add → Existing Item**. At "Add Existing Item" dialog, navigate to **Source\Assets** folder, select CloudBackgroundMobile.jpg and click **Add** button.

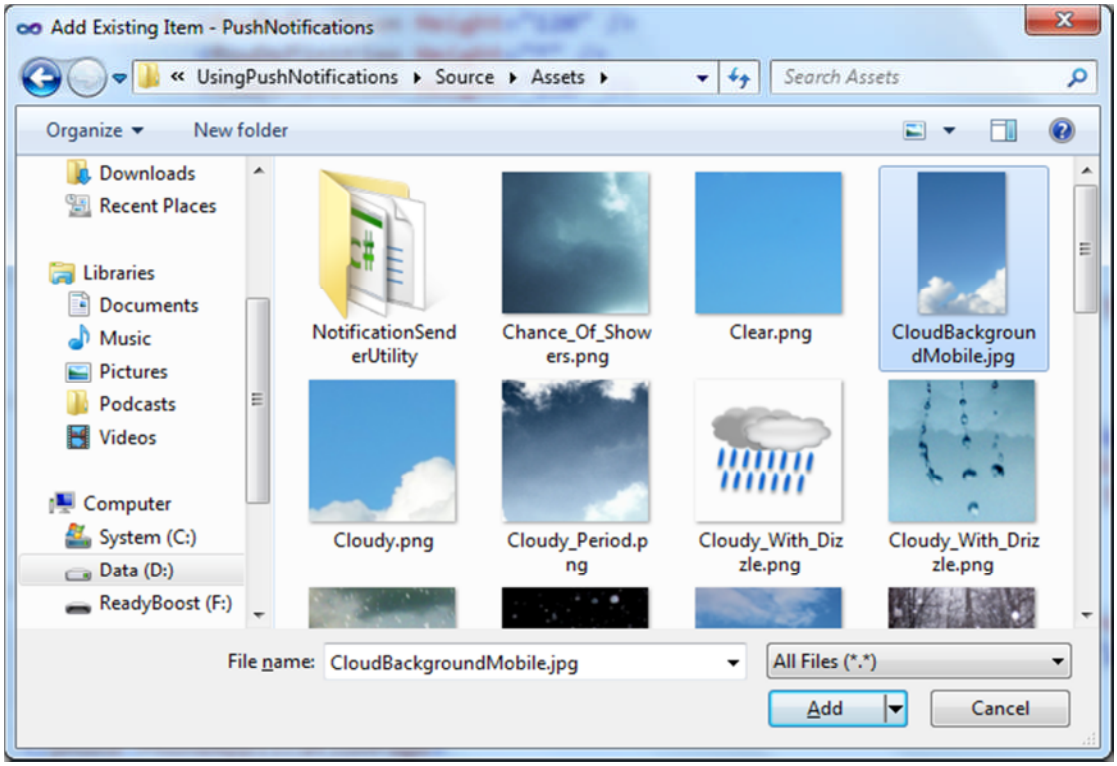


Figure 18

Adding existing image resource

7. Open the **MainPage.xaml** (if not opened automatically).
8. Locate the **LayoutRoot** grid and replace the content with the following code snippet:

XAML

```
<Grid x:Name="LayoutRoot" Background="Transparent">
   <Grid.RowDefinitions>
      <RowDefinition Height="120"/>
      <RowDefinition Height="*"/>
      <RowDefinition Height="150"/>
      <RowDefinition Height="Auto"/>
   </Grid.RowDefinitions>

   <Image Source="cloudbackgroundmobile.jpg" Grid.RowSpan="4" />

        <Grid x:Name="TitleGrid" Grid.Row="0"
VerticalAlignment="Top">
            <TextBlock Text="WEATHER SERVICE"
x:Name="textBlockPageTitle" Style="{StaticResource
PhoneTextPageTitle1Style}" />
        </Grid>
```

```
        <Grid Grid.Row="1" x:Name="ContentPanel"
Background="#10000000">
            <TextBlock x:Name="textBlockListTitle"
FontFamily="Segoe WP Light" FontSize="108" Text="City"
Margin="20,10,0,0" />
            <TextBlock x:Name="txtTemperature" FontFamily="Segoe
WP" FontSize="160" Text="80°" Margin="20,100,0,0" />
            <Image x:Name="imgWeatherConditions" Width="128"
Height="128" Stretch="None" HorizontalAlignment="Right"
VerticalAlignment="Top" Margin="20,155,20,0" />
        </Grid>

        <StackPanel Grid.Row="3" x:Name="StatusStackPanel"
Margin="20">
            <TextBlock FontSize="34" FontFamily="Segoe WP
Semibold" Foreground="#104f6f" Text="Status"
Style="{StaticResource PhoneTextNormalStyle}" />
            <TextBlock x:Name="txtStatus" FontFamily="Segoe WP"
FontSize="24" Foreground="#0a364c" Margin="0,0,0,0"
Style="{StaticResource PhoneTextNormalStyle}" Text="Not
Connected" TextWrapping="Wrap" />
        </StackPanel>

</Grid>
```

9. Open **MainPage.xaml.cs**.
10. Add the following using statements:

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

```
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;
using System.Collections.ObjectModel;
```

11. Add the following code snippet with private variables and constants to the beginning of the class:

    (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 = 30;
```

12. Add the following code snippet with helper functions. Those functions will update Windows Phone 7 application's UI status line and will print useful trace information:

    (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
```

13. Set the **PushNotifications** project as a startup project. To do it right-click on the project name and select **Set as Startup Project**.

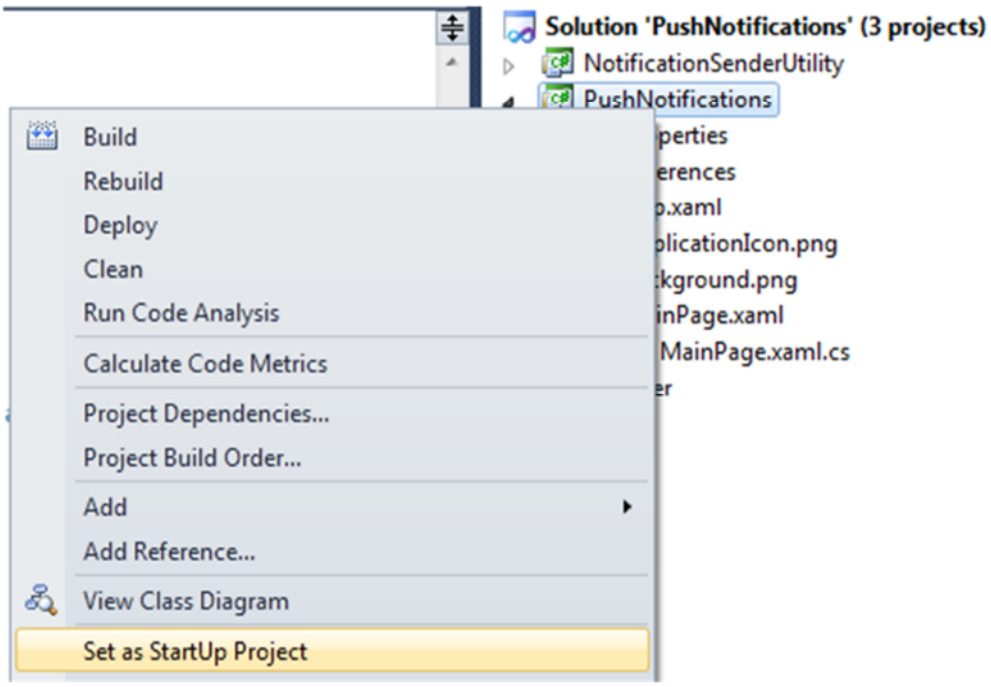


Figure 19

Set PushNotification as StartUp Project

14. Compile and run the application.
15. At this stage application looks like the following:

Figure 20
Running the Weather client

16. Stop the debugging and return to the code. This step concludes the current task.

Task 3 – Creating Notification Channel

During next couple steps you will create the functionality that is required to create a new push notification channel and subscribe to the channel events. The **HttpNotificationChannel** is a class, which creates a notification channel between the Push Notification service and the Push Client and creates a new subscription for raw, tile, and toast notifications. The channel creation flows goes like this: if the channel already exists, then client application should try to re-open it. Trying to re-create exists

channel will lead to exception. If the cannel is still not opened, then subscribe to the channel events and try to open the channel. Once channel will open it will fire **ChannelUriUpdated** event. This event could signal the client, that channel created successfully. Existing channel could be found by its name. In case of success in finding the channel by name it will be reactivated and could be used in the application. The entire process is asynchronous.

1. Open **MainPage.xaml.cs** from the **PushNotifications** project (if closed).

   Note: In current version of the emulator exists a bug, which causes an exception while trying to open the new channel right after the emulator's start-up. This bug prevents the emulator to open a valid channel for 20-30 seconds, and occurs every time the emulator is started. As a workaround this issue, this lab will check for the specific exception and will delay application execution for 30 seconds. After this time will run out it is safe to stop the application and run it again. ***Please do not shut down the emulator between application launches***.

2. Create a new region for miscellaneous functions according to the following code snippet:

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

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

3. Create Try/Catch blocks in the **DoConnect** function body according to the following code snippet:

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

```
C#
try
{

}
catch (Exception ex)
```

```
{
   Dispatcher.BeginInvoke(() => UpdateStatus("Channel error: " +
ex.Message));
}
```

4. Next, initialize a channel variable, subscribe to the channel events and try to open the channel. Create the **try** block body according to the following code snippet:

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

```
C#
//First, try to pick up existing channel
httpChannel = HttpNotificationChannel.Find(channelName);

if (null != httpChannel)
{
   Trace("Channel Exists - no need to create a new one");
   SubscribeToChannelEvents();

   Trace("Register the URI with 3rd party web service");
   SubscribeToService();

   //TODO: Place Notification

   Dispatcher.BeginInvoke(() => UpdateStatus("Channel
recovered"));
}
else
{
   Trace("Trying to create a new channel...");
   //Create the channel
   httpChannel = new HttpNotificationChannel(channelName,
"HOLWeatherService");
   Trace("New Push Notification channel created successfully");

   SubscribeToChannelEvents();

   Trace("Trying to open the channel");
   httpChannel.Open();
   Dispatcher.BeginInvoke(() => UpdateStatus("Channel open
requested"));
}
```

5. Create a helper function, which will do the subscription to the channel events:

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

C#

```
#region Subscriptions
private void SubscribeToChannelEvents()
{
   //Register to UriUpdated event - occurs when channel
successfully opens
   httpChannel.ChannelUriUpdated += new
EventHandler<NotificationChannelUriEventArgs>(httpChannel_Channel
UriUpdated);

   //Subscribed to Raw Notification
   httpChannel.HttpNotificationReceived += new
EventHandler<HttpNotificationEventArgs>(httpChannel_HttpNotificat
ionReceived);

   //general error handling for push channel
   httpChannel.ErrorOccurred += new
EventHandler<NotificationChannelErrorEventArgs>(httpChannel_Excep
tionOccurred);
}
#endregion
```

6. Add a helper function, which subscribes the Windows Phone 7 client to the Push Notification Service messages. The subscription will be done via registering with received channel URI to the registration service created in previous tasks. This function uses WebClient and hardcoded (in our case) RESTful WCF service URL to register the **client's URI** (received from the push server). Add the following code snippet to the **Subscriptions** region inserted previously:

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

   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));
}
```

7. Create the event handler function to handle **ChannelUriUpdate** event:

   (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
```

8. Add the **ExceptionOccured** event handler function to the region:

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

```
C#
void httpChannel_ExceptionOccurred(object sender,
NotificationChannelErrorEventArgs e)
{
   Dispatcher.BeginInvoke(() => UpdateStatus(e.ErrorType + "
occurred: " + e.Message));
}
```

9. Finally add the **HttpNotificationReceived** event handler function in the same region:

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

   This function should parse the payload XML received from Push Notification Service and update the UI of the client application. You will create this logic in next task.

10. Add a call to **DoConnect** at the class constructor. Resulting code of the constructor should look like the following code snippet:

```
C#
public MainPage()
{
   InitializeComponent();
   DoConnect();
}
```

11. Compile the application and fix compilation errors (if any).

12. Define multiple startup projects for this solution in order to run WPF Push Notification Client and Windows Phone 7 Push client together. In order to do this, in Solution Explorer right-click the solution name and select **Properties** from context menu:

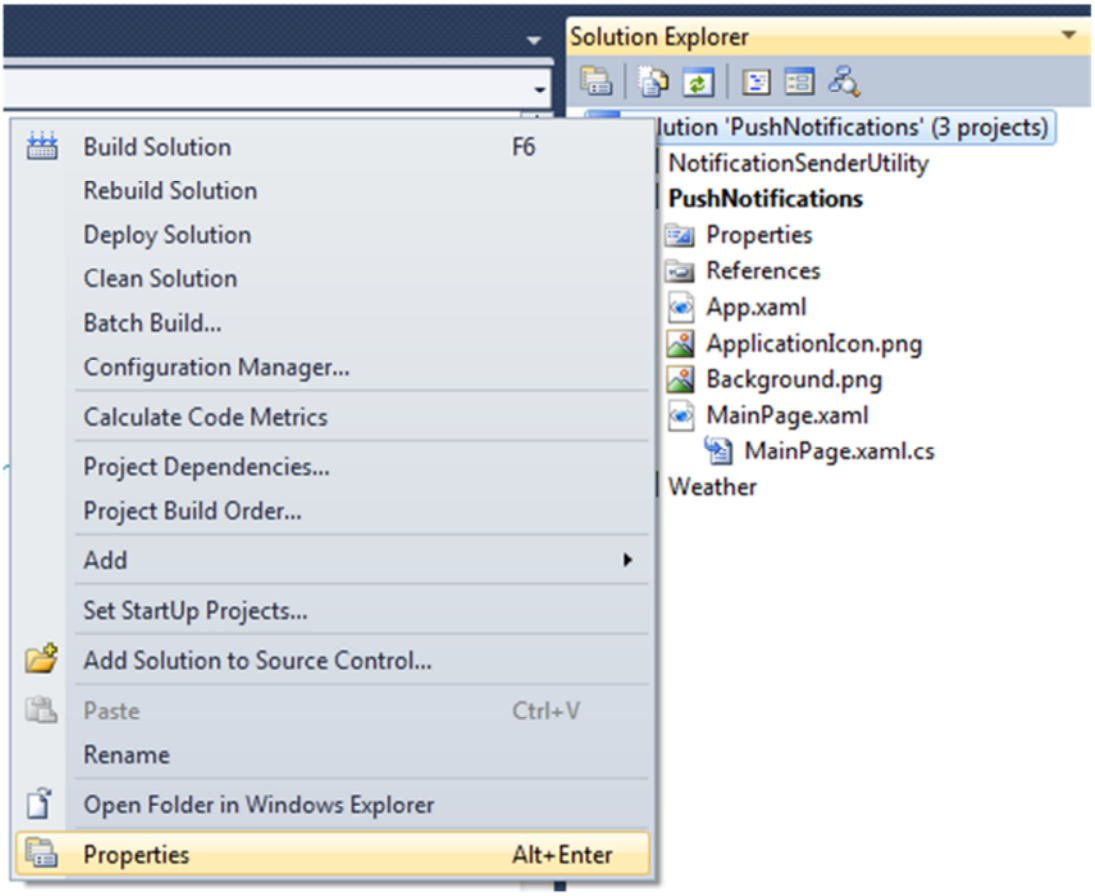


Figure 21

Opening Solution Properties

13. Select the **Startup Projects** page from **Common Properties** (if not selected automatically), select **Multiple startup projects** and set the **PushNotifications** and **Weather** projects as **Start** from the **Action** drop-down list:

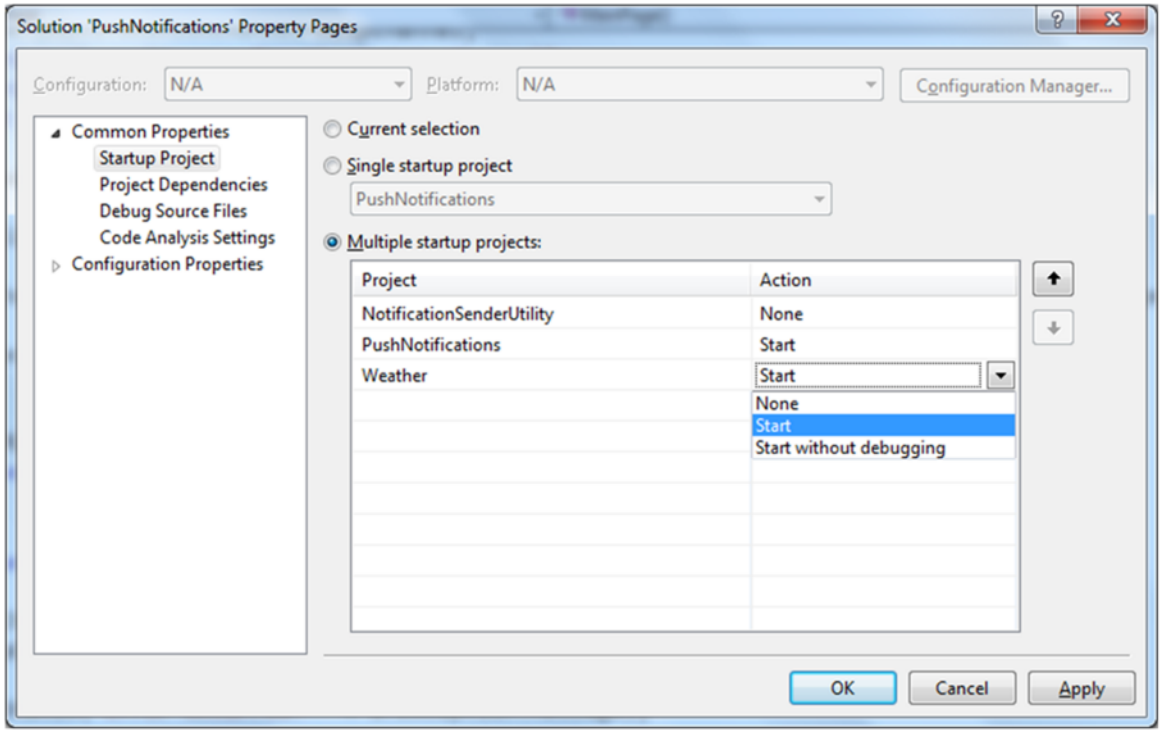


Figure 22

Selecting multiple startup projects

14. Press **F5** to compile and run the application.

15. After all projects start you screen should look like the follows:

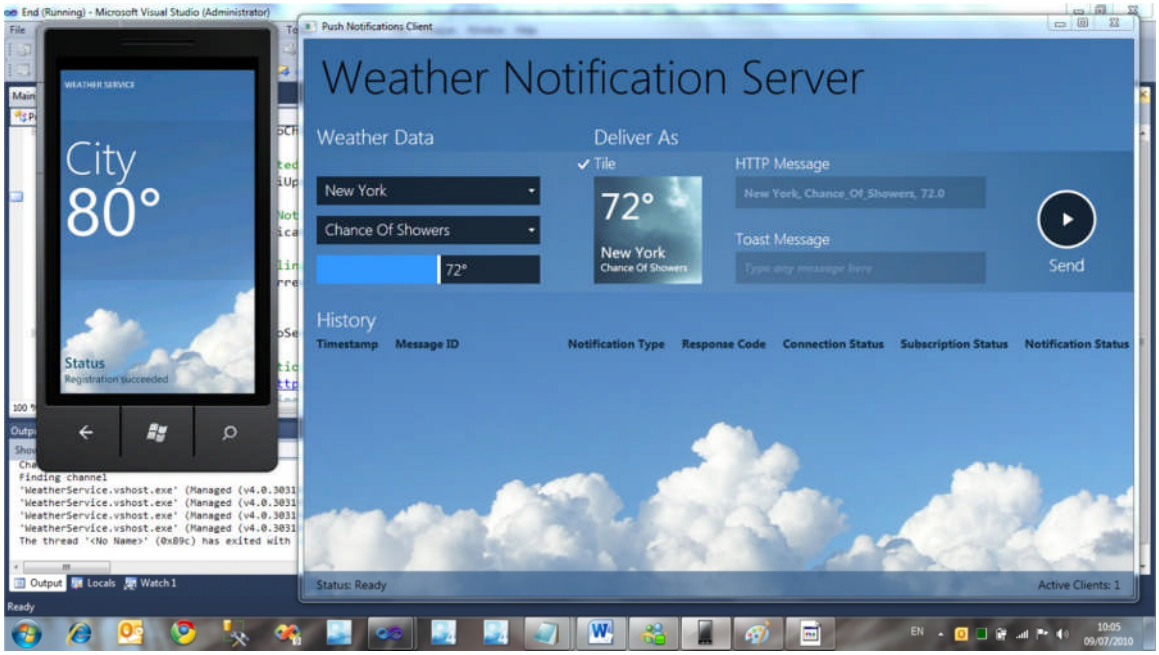


Figure 23

Running multiple projects

16. Make sure the Windows Phone 7 client application successfully registers with the Registration Service:

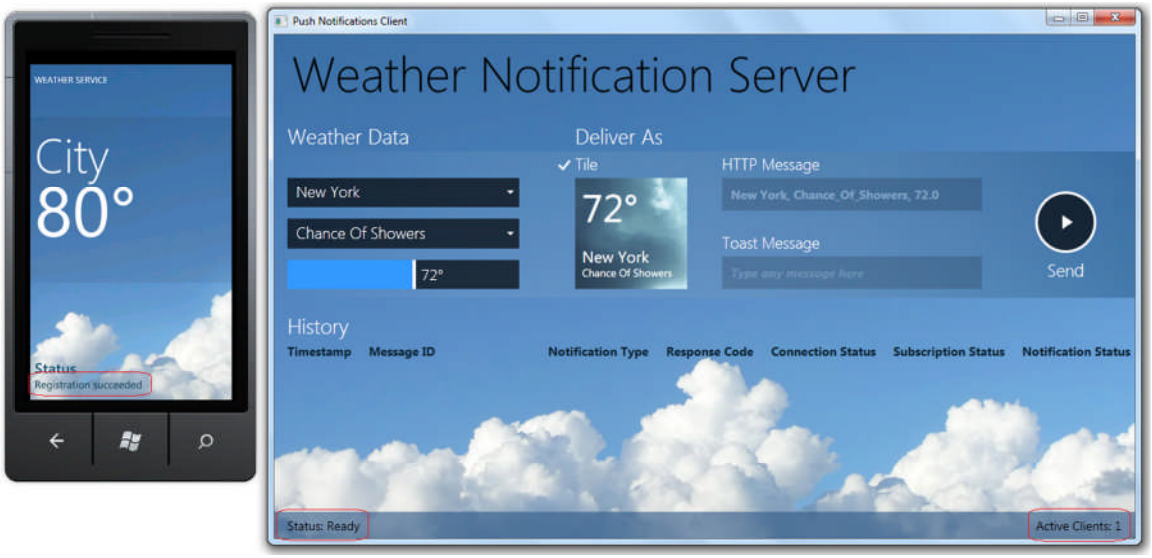


Figure 24

Checking client application registration

17. Put the breakpoint at **httpChannel_HttpNotificationReceived** function.

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

Figure 25

Breakpoint in notification event handler function

18. Change some parameters on the WPF Push Notification Client and click the **Send Http** button. When the breakpoint hit and execution of the code suspended, examine received information.

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

Figure 26

Breakpoint hit when notification arrives

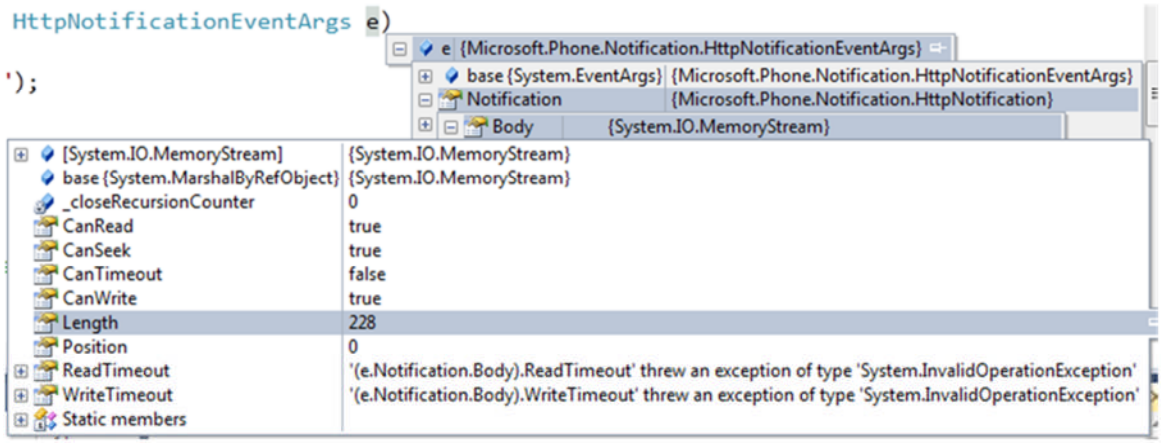


Figure 27

Notification payload received by Windows Phone 7 client application

19. Press **F5** to continue with program execution. Observe the new log entry in WPF Push Notification Client.

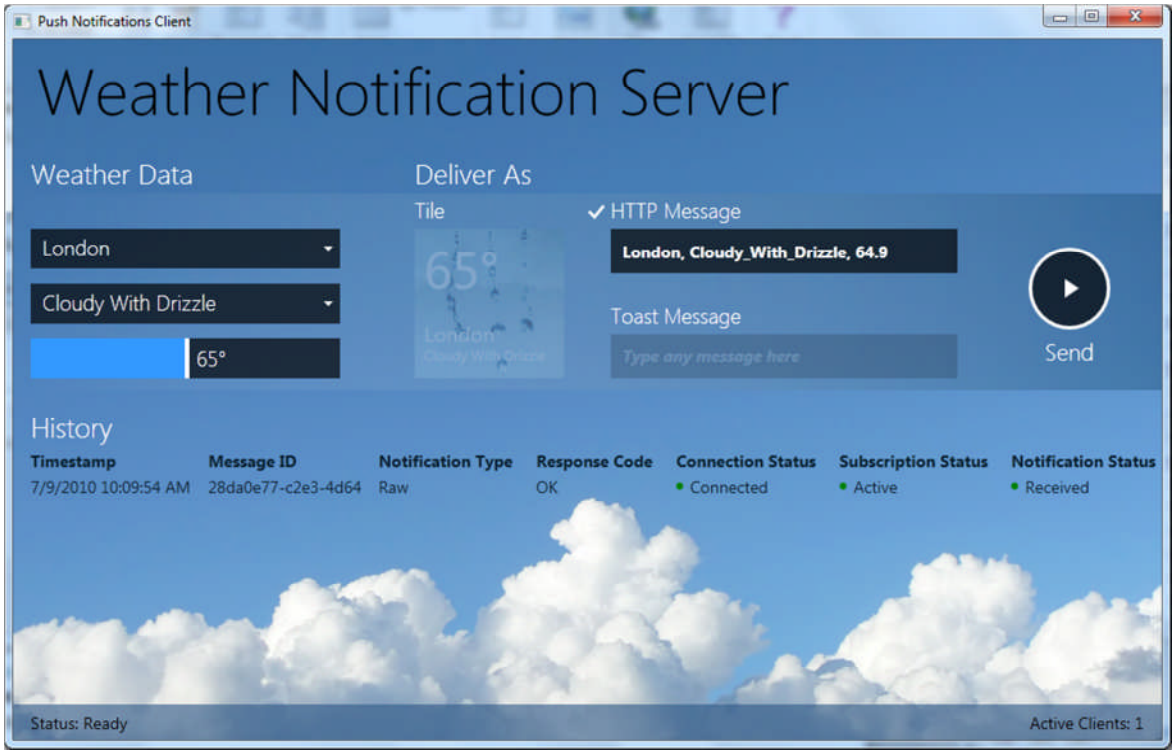


Figure 28

WPF Push Notification Client with updated log

20. Stop the debugging and return to the Visual Studio (***do not close Windows Phone 7 emulator!***). This step concludes the task.

Task 4 – Receiving and Processing Events from Push Notification Service

1. During this task you will create a function to parse XML arrived in the payload and update Windows Phone 7 application's UI. Add the following function to the **Misc logic** region of **MainPage.xaml.cs**:

   (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. In order to present weather graphically you need to add weather condition icons (provided as assets to this lab). In **PushNotifications** project create a new project folder and name it **Images**.

3. Add all existing PNG images (all images except CloudBackgroundMobile.jpg) from the Lab **Assets** folder (located under the **Source\Assets** folder of this lab*):*

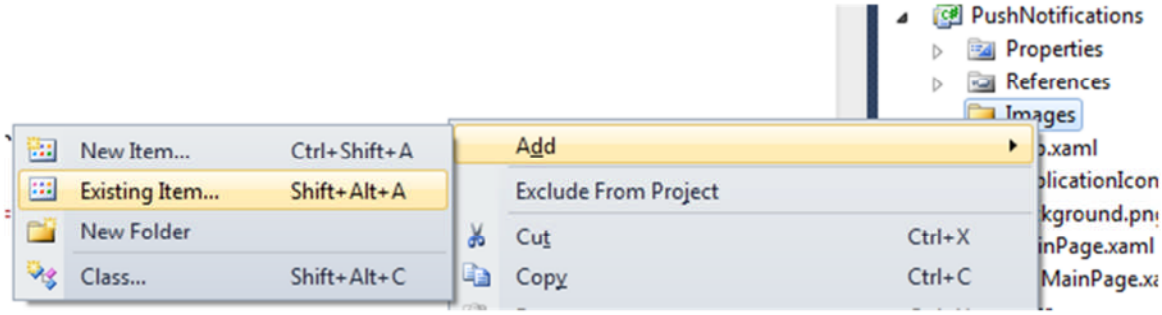


Figure 29

Adding existing images from Assets folder

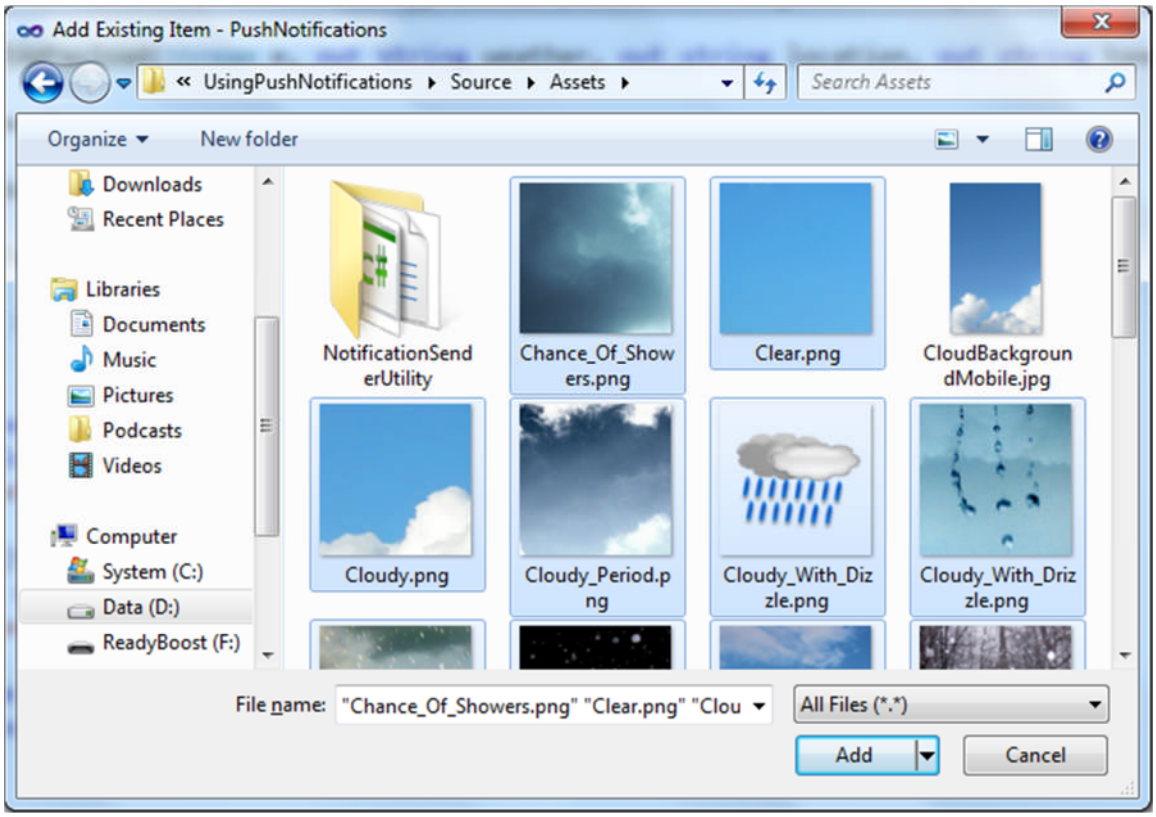


Figure 30

Adding existing images from Assets folder

4. Mark all the added images as **Content** in their build action. To do it open image **Properties** and select **Content** from the **Build Action** drop-down list.

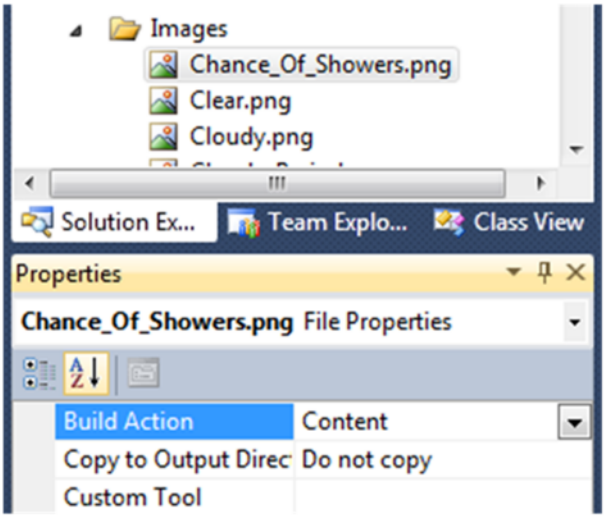


Figure 31

Marking image as Content resource

5. Recall the last function you created in previous task (**httpChannel_HttpNotificationReceived**, in MainPage.xaml.cs). This function will call to the parsing functionality created before and will update the UI. Add the following blue-highlighted code snippet after the "*//TODO - add parsing and UI updating logic here*" comment:

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

C#

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

    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));

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

6. Compile and run the application. After both application starts, change some values in WPF Push Notification Client and click **Send Http** button. Observe the notification arrives to the Windows Phone 7 client and UI changes. Observe trace information in Visual Studio 2010 **Output** window (if the Output window is not shown, click **Debug** | **Windows** | **Output** menu items or Ctrl+W, O).

Figure 32

Http Notification arrived and parsed

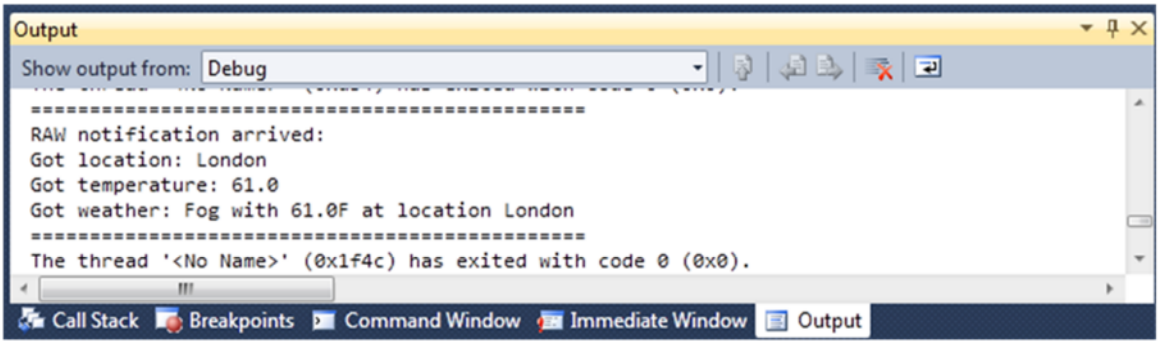


Figure 33

Trace information in Output window

7. Send the different notifications and observe the UI changes.

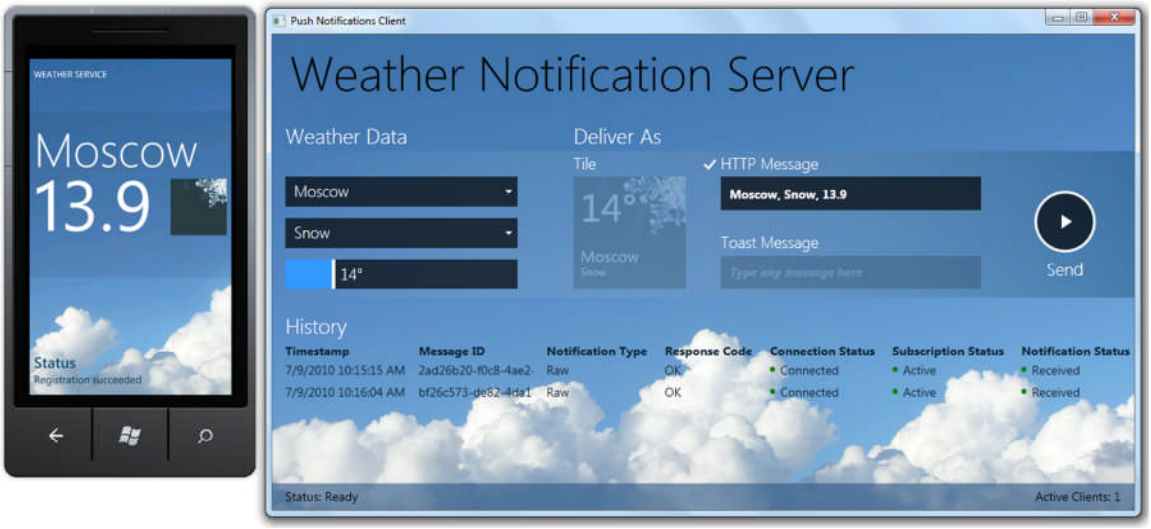


Figure 34

Http Notification arrived and parsed. Trace log updated

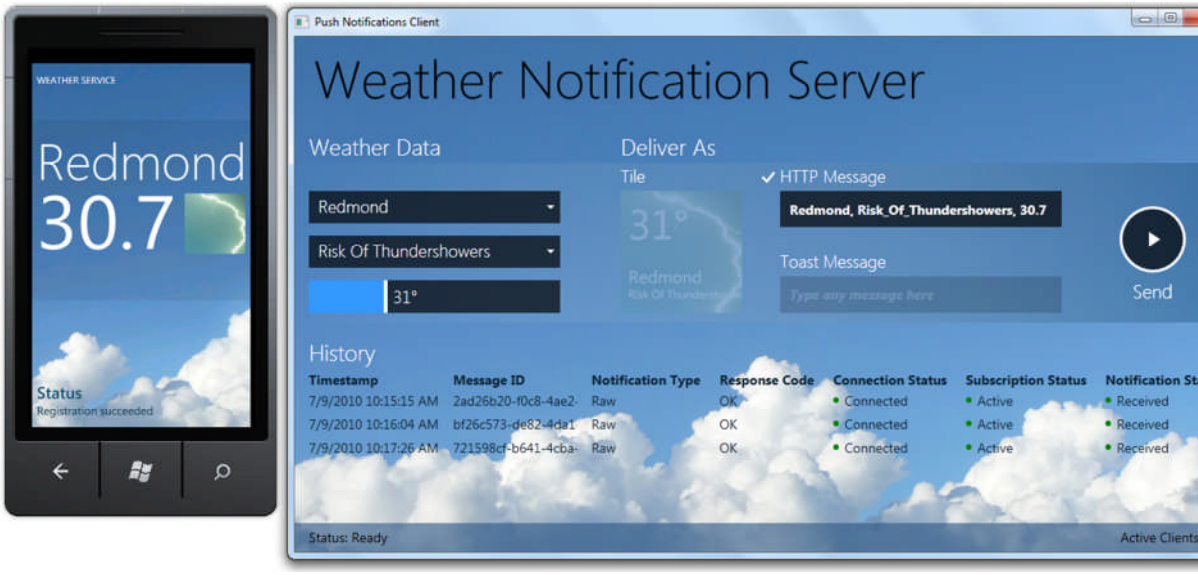


Figure 35

Http Notification arrived and parsed. Trace log updated

8. In Visual Studio, Press **SHIFT**+**F5** to stop the debugging and return to edit mode.
9. As mentioned before, application should try to recover existing channel. In order to do so, first the channel should be saved after successful open and then found by name in case it was saved before while application loads. Add the following code snippet to **MainPage.xaml.cs** – it contains a function to save and load channel information to Windows Phone 7 application's IsolatedStorage (for more information about Silverlight application's Isolated Storage refer to the following article: 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. Change the class constructor to call the loading functionality according to the following code snippet:

```
C#
public MainPage()
{
    InitializeComponent();
    if (!TryFindChannel())
        DoConnect();
}
```

11. Add the saving channel function call to the **httpChannel_ChannelUriUpdate** function. Add the following code snippet right after tracing *"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. Compile and run the applications twice. Use debugging to check that first time application does the channel opening and saves the channel info and the second time finds the channel by name.

This step concludes the exercise. During this exercise you learned how to communicate with Microsoft Push Notification Services, how to prepare and send the message to client Windows Phone 7 application and how to receive those messages on the Windows Phone 7.

> **Note:** The complete solution for this exercise is provided at the following location: **Source\Ex1-RawNotifications\End**.

Exercise 2: Introduction to the Toast and Tile Notifications for Alerts

In this section you will learn about two additional notification types – Toast and Tile notification. This lab covers the backend server that posts data to MSPNS, as well as how to register and handle these events on your Windows Phone 7 Application.

Tiles and toast notifications are two mechanisms that enable a cloud service to deliver relevant, actionable feedback to users outside an application's own user interface. Additionally, a cloud service can send a raw notification request. Depending on the type of notification sent, the notification will be routed either to the application or the shell.

Tile Notifications

A tile is a visual, dynamic representation of an application or its content within the Quick Launch area of the phone's Start experience. For example, a weather application may choose to display the user's local time and climate conditions in a tile. Because a cloud service can alter its tile's appearance at any time, this mechanism can be used to communicate information to the user on an ongoing basis. Each application that the user can launch on the phone is associated with a single tile, but only the user can control which of these tiles are pinned to the Quick Launch area.

A cloud service can control a tile's background image, counter (or 'badge'), and title properties. These properties are configured using the Windows Phone Developer Tools. Animation and sound properties are controlled by how the platform is configured, not by the application. For example, if the platform is configured to animate and beep upon any tile update, that is what will occur for any tile.

A tile's background image can reference either a local resource, which is part of the application deployment, or a cloud resource. By referencing a resource in the cloud, applications are enabled to dynamically update a tile's background image. This enables scenarios which require processing of the background image before it is displayed. In most scenarios, the application package should include all needed background images for the tile, since this is the best solution for performance and battery life.

Toast Notifications

A cloud service can generate a special kind of push notification known as a toast notification, which displays as an overlay onto the user's current screen. For example, a weather application may wish to display a toast notification if a severe weather alert is

in effect. If the user decides to click the toast notification, the application can launch and perform other actions.

A cloud service can control a toast notification's title and sub-title. The toast notification will also display the application's icon that is included in the application's deployment package.

Best Practices

- Toast notifications should be personally relevant and time critical.
- Toast notifications should primarily be focused on peer-to-peer communication.

Task 1 – Implementing Server Side of Sending Tiles & Toasts

1. Open Microsoft Visual Studio 2010 Express for Windows Phone from **Start | All Programs | Microsoft Visual Studio 2010 Express | Microsoft Visual Studio 2010 Express for Windows Phone**.

   > **Visual Studio 2010**: Open Visual Studio 2010 from **Start | All Programs | Microsoft Visual Studio 2010**.
   >
   > **Important note**: In order to run self-hosted WCF services within Visual Phone 2010 Express for Windows Phone or Microsoft Visual Studio 2010 it should be opened in Administrative Mode. For reference about creating and hosting self-hosted WCF services see MSDN article (http://msdn.microsoft.com/en-us/library/ms731758.aspx). In order to open Visual Studio 2010 Express for Windows Phone or Visual Studio 2010 in Administrative Mode locate the Microsoft Visual Studio 2010 Express for Windows Phone shortcut at **Start | All Programs | Microsoft Visual Studio 2010 Express** or Microsoft Visual Studio 2010 shortcut at **Start | All Programs | Microsoft Visual Studio 2010**, right-click on the icon and select "Run as administrator" from the opened context menu. The UAC notification will pop up. Click "Yes" in order to allow running the Visual Studio 2010 Express for Windows Phone or Visual Studio 2010 with elevated permissions.

2. Open the **Begin.sln** starter solution from the **Source\Ex2-TileToastNotifications\Begin** folder of this lab. Alternatively, you can continue working on the solution created in previous exercise.
3. Open the **NotificationSenderUtility.cs** file located under the **NotificationSenderUtility** project.
4. Toast and Tile notification are system defined notification in Windows Phone 7 platform. This is different from RAW notifications, where all applications could create their own payload format and parse it accordingly. During next couple

steps you will create general functionality to create Tile & Toast messages payload – this payload will work for any Windows Phone 7 application. In addition you will expose public functionality to send such messages and connect WPF Push Notification Client's buttons events to it. Locate the **SendXXXNotification functionality** region. Add the following public functions to the region (they will be used by the WPF application later):

(Code Snippet – *Using Push Notifications – NotificationSenderUtility – SendXXXNotification functions*)

C#

```
public void SendToastNotification(List<Uri> Uris, string
message1, string message2, SendNotificationToMPNSCompleted
callback)
{
   byte[] payload = prepareToastPayload(message1, message2);

   foreach (var uri in Uris)
      SendNotificationByType(uri, payload,
NotificationType.Toast, callback);
}

public void SendTileNotification(List<Uri> Uris, string TokenID,
string BackgroundImageUri, int Count, string Title,
SendNotificationToMPNSCompleted callback)
{
   byte[] payload = prepareTilePayload(TokenID,
BackgroundImageUri, Count, Title);

   foreach (var uri in Uris)
      SendNotificationByType(uri, payload,
NotificationType.Token, callback);
}
```

5. Next you'll create a new region with two functions – first to prepare the Toast notification payload, and second for Tile notification payload preparation.

   Tile notification message should be in the following format. Notice that the *<background image path>, <count>* and *<title>* elements are in a string format.

XML

```
Content-Type: text/xml
X-WindowsPhone-Target: token

<?xml version="1.0" encoding="utf-8"?>
<wp:Notification xmlns:wp="WPNotification">
```

```
   <wp:Tile>
      <wp:BackgroundImage><background image
path></wp:BackgroundImage>
      <wp:Count><count></wp:Count>
      <wp:Title><title></wp:Title>
   </wp:Tile>
</wp:Notification>
```

Toast notification message from the other side should be in the following format. Notice that *<Text1>* and *<Text2>* are in string format.

XML

```
Content-Type: text/xml
X-WindowsPhone-Target: toast

<?xml version="1.0" encoding="utf-8"?>
<wp:Notification xmlns:wp="WPNotification">
   <wp:Toast>
      <wp:Text1><string></wp:Text1>
      <wp:Text2><string></wp:Text2>
   </wp:Toast>
</wp:Notification>
```

6. Create the **Prepare Payload** family of functions according to the following code snippet:

   (Code Snippet – *Using Push Notifications – NotificationSenderUtility – Prepare Payload functions*)

C#

```
#region Prepare Payloads
private static byte[] prepareToastPayload(string text1, string
text2)
{
    MemoryStream stream = new MemoryStream();
    XmlWriterSettings settings = new XmlWriterSettings() { Indent
= true, Encoding = Encoding.UTF8 };
    XmlWriter writer = XmlWriter.Create(stream, settings);
    writer.WriteStartDocument();
    writer.WriteStartElement("wp", "Notification",
"WPNotification");
    writer.WriteStartElement("wp", "Toast", "WPNotification");
    writer.WriteStartElement("wp", "Text1", "WPNotification");
    writer.WriteValue(text1);
    writer.WriteEndElement();
    writer.WriteStartElement("wp", "Text2", "WPNotification");
```

```
    writer.WriteValue(text2);
    writer.WriteEndElement();
    writer.WriteEndElement();
    writer.WriteEndDocument();
    writer.Close();

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

private static byte[] prepareTilePayload(string tokenId, string
backgroundImageUri, int count, string title)
{
    MemoryStream stream = new MemoryStream();
    XmlWriterSettings settings = new XmlWriterSettings() { Indent
= true, Encoding = Encoding.UTF8 };
    XmlWriter writer = XmlWriter.Create(stream, settings);
    writer.WriteStartDocument();
    writer.WriteStartElement("wp", "Notification",
"WPNotification");
    writer.WriteStartElement("wp", "Tile", "WPNotification");
    writer.WriteStartElement("wp", "BackgroundImage",
"WPNotification");
    writer.WriteValue(backgroundImageUri);
    writer.WriteEndElement();
    writer.WriteStartElement("wp", "Count", "WPNotification");
    writer.WriteValue(count.ToString());
    writer.WriteEndElement();
    writer.WriteStartElement("wp", "Title", "WPNotification");
    writer.WriteValue(title);
    writer.WriteEndElement();
    writer.WriteEndElement();
    writer.Close();

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

#endregion
```

7. Open **MainWindow.xaml.cs** from the **Weather** project.
8. Locate the **sendToast** function. This function should get the Toast message from the Push Notification Client UI and send it to all the subscribers. Add the following code snippet in the function body:

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

```
C#
private void sendToast()
{
    string msg = txtToastMessage.Text;
    txtToastMessage.Text = "";
    List<Uri> subscribers = RegistrationService.GetSubscribers();
    ThreadPool.QueueUserWorkItem((unused) =>
notifier.SendToastNotification(subscribers,
        "WEATHER ALERT", msg, OnMessageSent));
}
```

9. Locate the **sendTile** function. This function should get the parameters from Push Notification Client UI and send it to all the subscribers. Add the following code snippet in the function body:

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

```
C#
private void sendTile()
{
    string weatherType = cmbWeather.SelectedValue as string;
    int temperature = (int)sld.Value;
    string location = cmbLocation.SelectedValue as string;
    List<Uri> subscribers = RegistrationService.GetSubscribers();
    ThreadPool.QueueUserWorkItem((unused) =>
notifier.SendTileNotification(subscribers,
"PushNotificationsToken", "/Images/" + weatherType + ".png",
temperature, location, OnMessageSent));
}
```

10. Compile and run the applications. Check that messages are dispatched to the Push Notification Service.

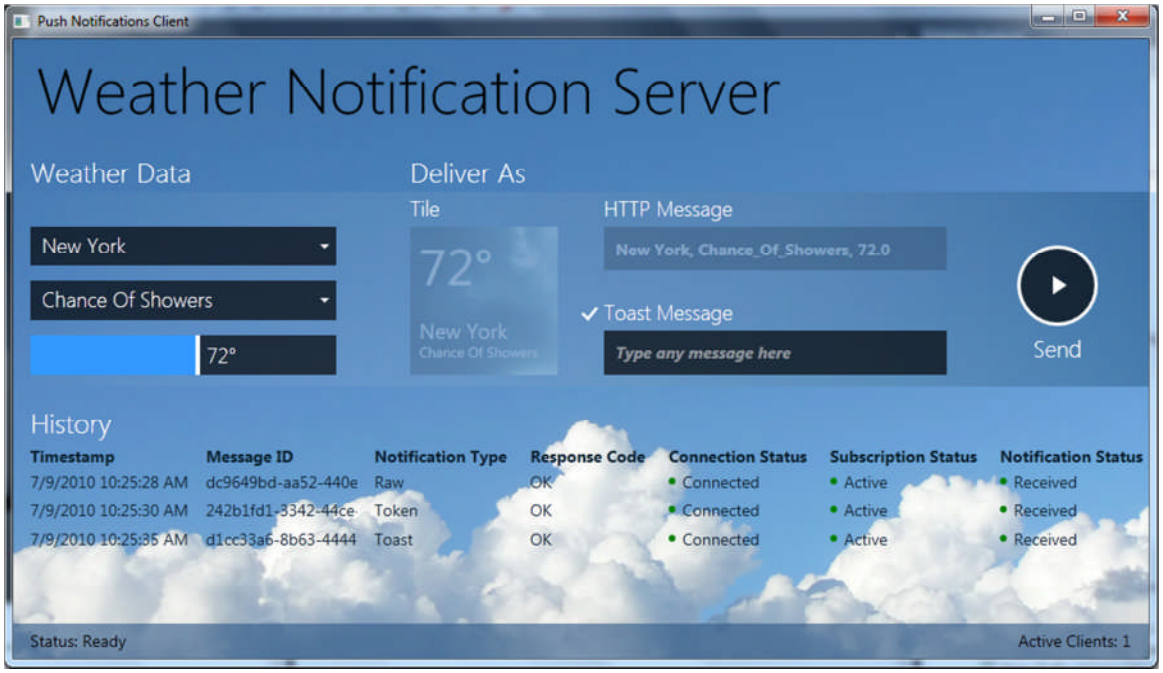
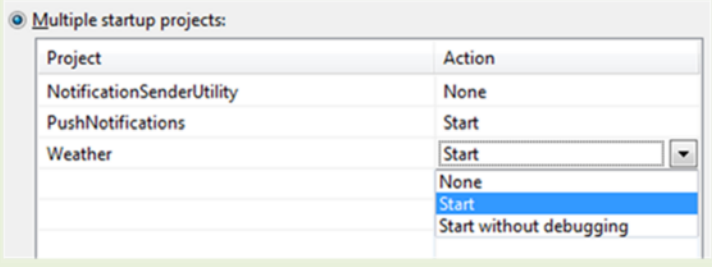


Figure 36

WPF Push Notifications Client log

Note: If you've started this exercise from the Begin solution instead of continuing with the previous exercise solution, to run the application, you first need to define multiple startup projects for this solution in order to run WPF Push Notification Client and Windows Phone 7 Push client together.

In order to do this, in Solution Explorer right-click on the solution name and select Properties from the context menu. Select the **Startup Projects** page from **Common Properties** (if not selected automatically), select **Multiple startup projects** and set the **PushNotifications** and **Weather** projects as **Start** from the **Action** drop-down list.



This step concludes the task.

Task 2 – Processing Tile & Toast Notifications on the Phone

1. Open **MainPage.xaml.cs** in the **PushNotifications** project.
2. In the next few steps you will subscribe to Tile and Toast notification events and will handle those events. Locate the **Subscriptions** region and add the following code snippet:

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

C#

```
private void SubscribeToNotifications()
{
    ////////////////////////////////////////
    // Bind to Toast Notification
    ////////////////////////////////////////
    try
    {
        if (httpChannel.IsShellToastBound == true)
        {
            Trace("Already bounded (register) to to Toast
notification");
        }
        else
        {
            Trace("Registering to Toast Notifications");
            httpChannel.BindToShellToast();
        }
    }
    catch (Exception ex)
    {
        // handle error here
    }

    ////////////////////////////////////////
    // Bind to Tile Notification
    ////////////////////////////////////////
    try
    {
        if (httpChannel.IsShellTileBound == true)
        {
            Trace("Already bounded (register) to Tile
Notifications");
        }
        else
        {
            Trace("Registering to Tile Notifications");
```

```
            // you can register the phone application to receive
tile images from remote servers [this is optional]
            Collection<Uri> uris = new Collection<Uri>();
            uris.Add(new
Uri("http://jquery.andreaseberhard.de/pngFix/pngtest.png"));

            httpChannel.BindToShellTile(uris);
        }
    }
    catch (Exception ex)
    {
        //handle error here
    }
}
```

3. Now locate the **SubscribeToChannelEvents** function and add the following blue-highlighted code snippet to the function body:

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

C#

```
private void SubscribeToChannelEvents()
{
    //Register to UriUpdated event - occurs when channel
successfully opens
   httpChannel.ChannelUriUpdated += new
EventHandler<NotificationChannelUriEventArgs>(httpChannel_Channel
UriUpdated);

   //Subscribed to Raw Notification
   httpChannel.HttpNotificationReceived += new
EventHandler<HttpNotificationEventArgs>(httpChannel_HttpNotificat
ionReceived);

   //general error handling for push channel
   httpChannel.ErrorOccurred += new
EventHandler<NotificationChannelErrorEventArgs>(httpChannel_Excep
tionOccurred);
   //subscrive to toast notification when running app
   httpChannel.ShellToastNotificationReceived += new
EventHandler<NotificationEventArgs>(httpChannel_ShellToastNotific
ationReceived);
}
```

4. Locate the **Channel event handlers** region and add the following function to handle the events:

   (Code Snippet – *Using Push Notifications – MainPage.xaml.cs – Tile and Toast notifications event handler function*)

```
C#
void httpChannel_ShellToastNotificationReceived(object sender,
NotificationEventArgs e)
{
   Trace("==============================================");
   Trace("Toast/Tile notification arrived:");
   foreach (var key in e.Collection.Keys)
   {
      string msg = e.Collection[key];

      Trace(msg);
      Dispatcher.BeginInvoke(() => UpdateStatus("Toast/Tile
message: " + msg));
   }

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

> **Note:** In our simple case the functions just tracing the message payload, but in real-world application you could use it to perform any business logic.

5. Lastly, add the following code snippet to the following number of locations:

   (Code Snippet – *Using Push Notifications – MainPage.xaml.cs – Subscribe toNotifications call*)

```
C#
SubscribeToNotifications();
```

   The locations to add this code snippet are the following:

   a. In the **DoConnect** function, in the **try** block, between "*SubscribeToService();*" and "*Dispatcher.BeginInvoke();*"

```
C#
if (null != httpChannel)
{
    Trace("Channel Exists - no need to create a new one");
    SubscribeToChannelEvents();

    Trace("Register the URI with 3rd party web service");
    SubscribeToService();
```

```
    Trace("Subscribe to the channel to Tile and Toast
notifications");
    SubscribeToNotifications();

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

b. In the **httpChannel_ChannelUriUpdated** function, between "*SubscribeToService();*" and "*Dispatcher.BeginInvoke(…);*"

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

6. Press **F5** to compile and run the applications. On the phone emulator, click the Back button (←) to exit the Push Notification application and go to the Quick Launch area.
7. In the Quick Launch area, click the Right Arrow button to get to the "All Applications" screen.

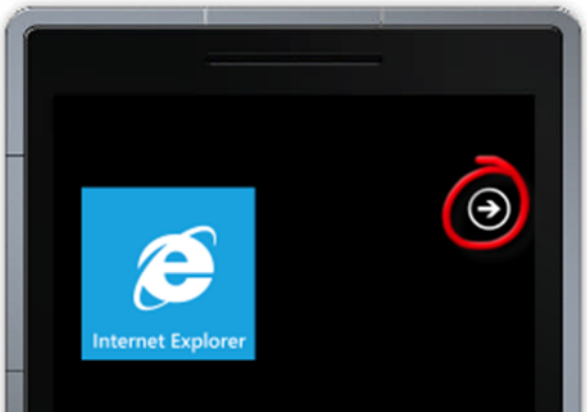


Figure 37

Opening the installed applications screen

8. Locate the **PushNotifications** item, push it down and hold until context menu pops-up. Click **Pin to Start**.

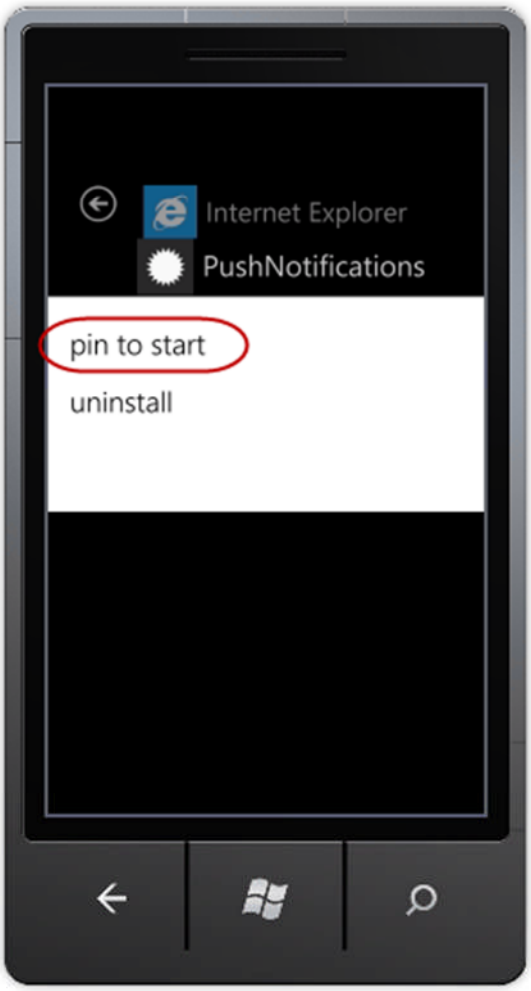


Figure 38
Pinning tile to Start screen

9. The emulator will automatically go back to the Quick Launch area where you'll notice the **PushNotifications** tile pinned.

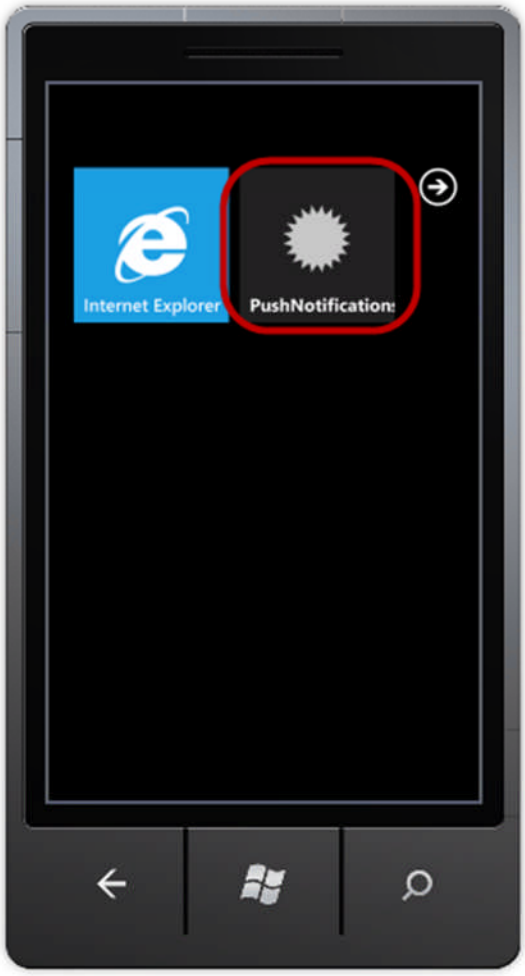


Figure 39

PushNotifications Tile

10. On WPF Push Notification Client, change some notifications parameters and click the **Send Tile** button. Observe the Tile change in emulator. If you click the *Toast message* you will be taken back to the application.

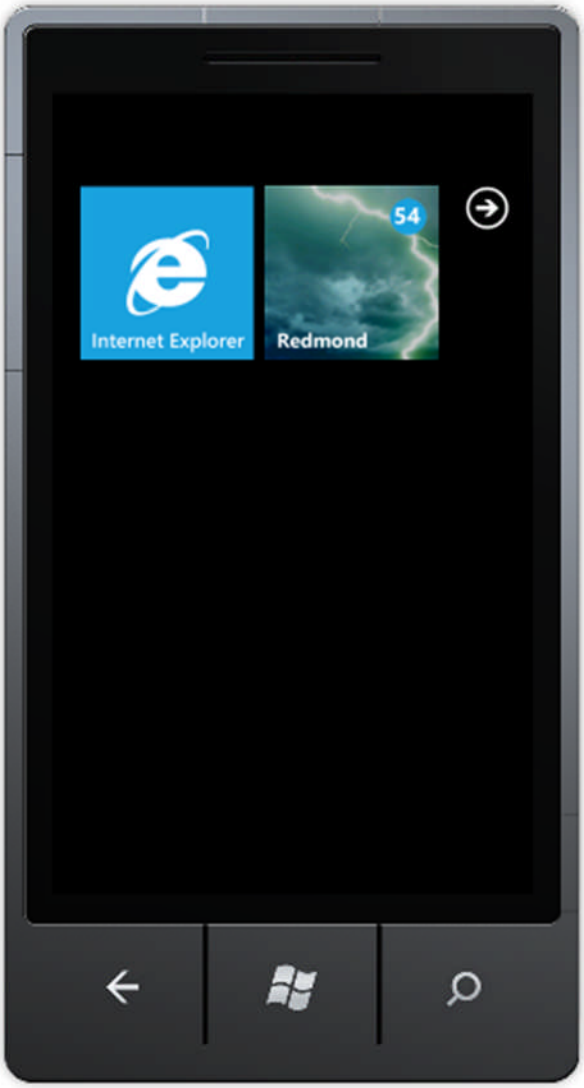


Figure 40

Tile Notification arrived on emulator

11. On the WPF Push Notification Client enter some message and click the **Send Toast** button. Observe how the Toast message arrives to the phone. Click the *Toast message* to switch back to the application.

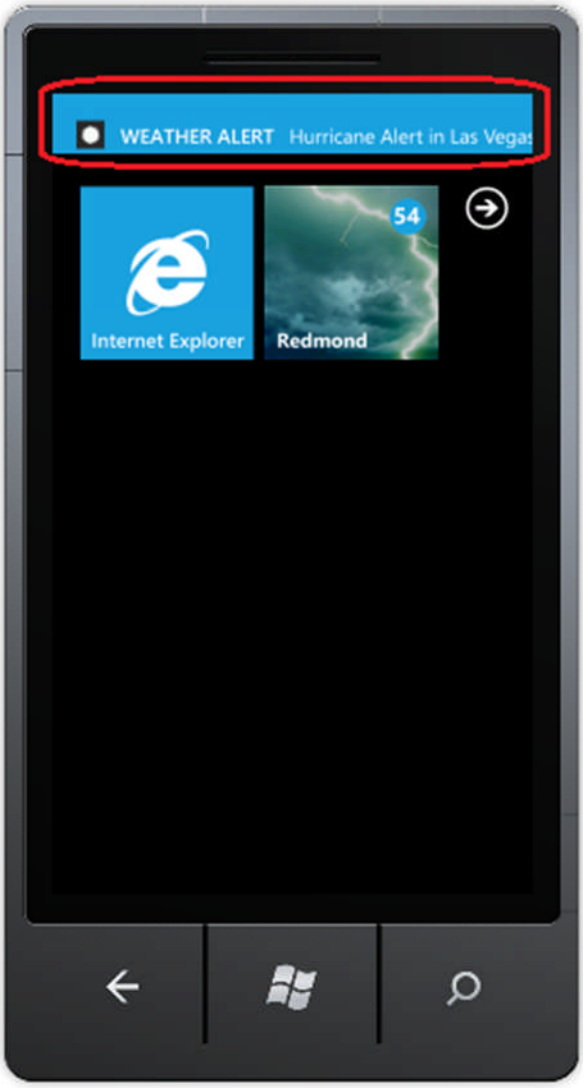


Figure 41

Toast message on phone emulator

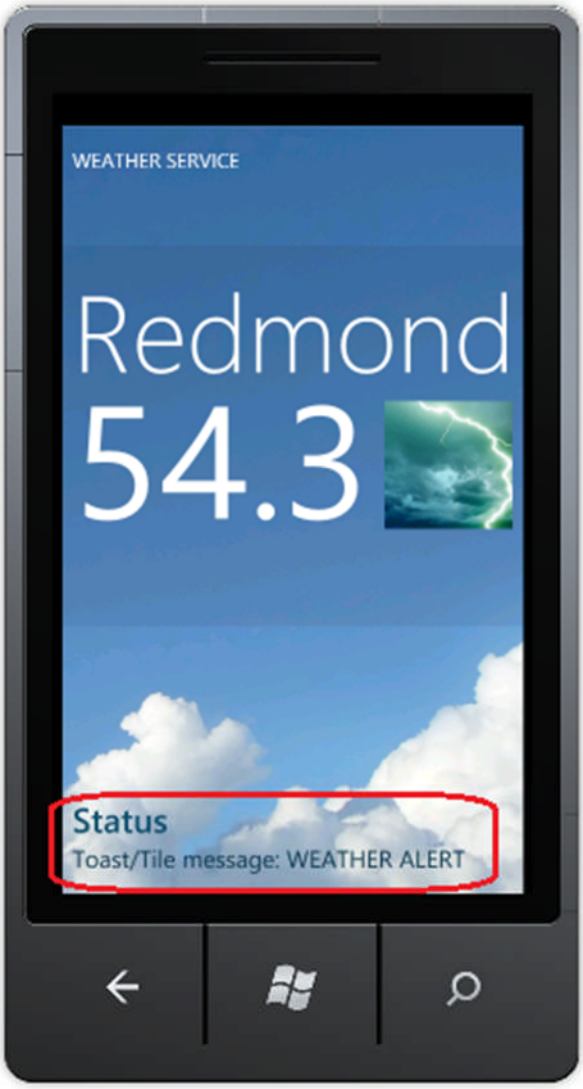


Figure 42

Toast message on Windows Phone Application

This step concludes the task.

Task 3 – Processing Scheduled Tile Notifications on the Phone

Note: You can also update the application's tile by using a shell tile schedule represented by the Microsoft.Phone.Shell.**ShellTileSchedule** class. This special class allows an application to schedule updates of its tile's background image by setting the background image fully qualified URI and its related scheduler recurrence and interval attributes. When the phone starts the tile schedule instance, it automatically sends a tile notification to the application that then fetches the image based on the qualified URI, and updates the tile.

In the next few steps you will create an instance of a ShellTileSchedule class that performs updates of the application tile under the hood.

1. Open App.xaml.cs in the PushNotifications project.
2. Locate the application's constructor - **App** method, and insert the following code fragment right after it:

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

C#

```
// To store the instance for the application lifetime
private ShellTileSchedule shellTileSchedule;

  /// <summary>
  /// Create the application shell tile schedule instance
  /// </summary>
  private void CreateShellTileSchedule()
  {
      shellTileSchedule = new ShellTileSchedule();
      shellTileSchedule.Recurrence = UpdateRecurrence.Interval;
      shellTileSchedule.Interval = UpdateInterval.EveryHour;
      shellTileSchedule.StartTime = DateTime.Now;
      shellTileSchedule.RemoteImageUri = new
Uri(@"http://cdn3.afterdawn.fi/news/small/windows-phone-7-
series.png");
      shellTileSchedule.Start();
  }
```

Note that you can only provide a **RemoteImageUri**. Therefore you must provide an online and available URI that represents an image to download and display. You can't reference URI from your local application. The image size can **NOT** exceed 80KB, and download time can **NOT** exceed 60 sec.

3. Now, locate the **App()** function and call the **CreateShellTileSchedule** function from it (see the highlighted line):

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

C#

```
// Constructor
public App()
{
```

```
    // Global handler for uncaught exceptions.
    // Note that exceptions thrown by ApplicationBarItem.Click
will not get caught here.
    UnhandledException += Application_UnhandledException;

    // Standard Silverlight initialization
    InitializeComponent();

    // Phone-specific initialization
    InitializePhoneApplication();

    // Create the shell tile schedule instance
    CreateShellTileSchedule();
}
```

You just provided the application with the object that will update the application tile according to the schedule properties.

You have set the property:

- **Recurrent** to the value **Interval**, which causes the tile to be periodically updated. The other possible value for this property is **OneTime**, which causes the tile to update only once.
- **Interval** to the value **EveryHour**, which causes the tile to be updated every 60 minutes. It is worth mentioning that an hour is the minimum possible value for the update interval. This restriction is in place to save phone resources.
- **RemoteImageUri** to the address where you want to get tile image updates from.

And after all that, you have called the Start method to start updating the tile. Be aware that the first update may delay for an hour because this class invokes updates on full hours. The tile update schedule is kind of funky. Even if you set StartTime to now, you'll need to wait for the phone update, which occurs up to every 60 minutes. Unfortunately, this is a system limitation, so if you want to debug this code, you have to wait for that one hour. I usually debug this code overnight.

4. Press F5 to compile and run the applications. On the phone emulator, click the Back button () to exit the Push Notification application and go to the Quick Launch area.

5. Verify that your application tile is pinned in the Quick Launch area, else pin it according to the procedure defined in steps 7-9 of Task 2.
6. The PushNotifications tile image now will be updated according to the ShellTileSchedule definitions. So when the first schedule interval expires sometime in the next hour, the application tile will display the image located at the address stored in RemoteImageUri property, http://cdn3.afterdawn.fi/news/small/windows-phone-7-series.png.

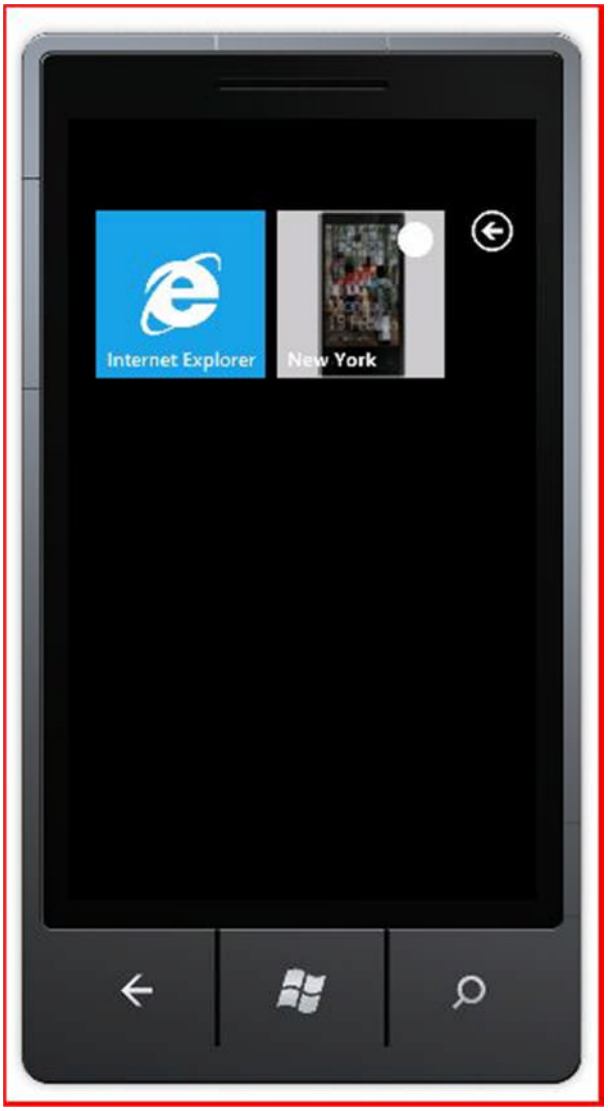


Figure 43
Scheduled Tile Image

This concludes the exercise and the lab.

Note: The complete solution for this exercise is provided at the following location: **Source\Ex2-TileToastNotifications\End**.

Summary

During this lab you learned about the notification services for the Windows Phone 7 platform. You learned about notifications types, and how to prepare and send them through Microsoft Push Notification Service. You created the business client application to prepare and send such messages and Windows Phone 7 client application. The Windows Phone 7 client application subscribes to the notifications and updates the UI according to the information received in the messages.